UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Marko Tavčar

Modularni sistem za avtomatsko vodenje stavb

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Ljubljana, 2015

UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Marko Tavčar

Modularni sistem za avtomatsko vodenje stavb

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: izr. prof. dr. Uroš Lotrič

Ljubljana, 2015

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Razvijte lasten sistem za vodenje in nadzor stavb, ki naj vključuje ustrezno strojno ter programsko opremo in vmesnik človek-stroj. Cenovno ugoden sistem naj bo zasnovan modularno in naj omogoča postopno uvajanje avtomatskega vodenja v posamezne dele stavbe.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Marko Tavčar sem avtor diplomskega dela z naslovom:

Modularni sistem za avtomatsko vodenje stavb

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom izr. prof. dr. Uroša Lotriča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 29. septembra 2015 Podpis avtorja:

Zahvalil bi se rad mentorju, izr. prof. dr. Urošu Lotriču za vložen trud in potrpljenje pri izdelavi tega diplomskega dela, družini, ki me je podpirala pri študiju in seveda sošolcem s katerimi smo skupaj premagovali ovire na poti do diplome.

Research is what I'm doing
when I don't know what I'm doing.

Warnher von Braun

Kazalo

Povzetek

V diplomskem delu bomo opisali sistem za krmiljenje objektov, v katerih želimo krmiliti najrazličnejše končne porabnike. Primarni elementi krmiljenja so luči in motorji senčil, sistemu pa bomo dodali tudi različne senzorje. Vmesnik človek-stroj za sistem bo zasnovan kot mobilna aplikacija za operacijski sistem Android. Predstavili bomo arhitekturo sistema in razloge zanjo, nato sledi opis uporabljene strojne opreme ter programskih rešitev. Od sistema želimo možnost postopne integracije v objekt ter enostavno nadgradnjo. Eden izmed glavnih ciljev je tudi primernost za vgradnjo v obstoječe objekte in čim večjo uporabo obstoječih elementov. Ocenili bomo tudi strošek za integracijo takšnega sistema v obstoječ objekt.

Ključne besede: pametna hiša, krmilni sistem, Arduino, Raspberry Pi, Xamarin, Android

Abstract

In this thesis, we are going to describe a modular system for control of buildings with a wide variety of actuators. We will primarily focus on control of lamps and motor shades, and add general support for all sorts of sensors too. Human-machine interface consist of mobile application for operating systems Android, which connects to the control system. We are going to present and explain the system architecture including the description of the hardware and software solutions. We designed the system is such a way that it enables progressive integration in buildings. One of the main goals of the system is the suitability for installation in existing buildings by maximizing the use of existing control elements. We also assessed the cost of introducing such a system in an existing building.

Keywords: smart house, control system, Arduino, Raspberry Pi, Xamarin, Android

Poglavje 1 Uvod

Predstavljajmo si, da smo podjetnik srednjih let in za nami je naporen dan, pol sestankov in drugih obveznosti. S svojim avtomobilom Mercedes-Benz se pripeljemo pred vhod v hišo in dovozna vrata odpremo s pomočjo pametnega telefona. Zapeljemo se na dovoz, parkiramo in se odpravimo v hišo. Utrujeni se usedemo na kavč in s pametnim telefonom zasenčimo prostor, ugasnemo luči, prižgemo televizor in si privoščimo zaslužen oddih.

Zgodbica je sicer izmišljena, vendar ni znanstvena fantastika. Take avtomatizirane hiše so že nekaj let povsem realno izvedljive. Ker pa vsi nimamo tako globokih žepov, gre vendarle za nekakšen pogled v prihodnost.

Področje avtomatizacije je zelo široko in sega od proizvodnje, letalske in avtomobilistične industrije do pametnih hiš [19]. Dandanes si proizvodnje v številnih panogah brez avtomatiziranih procesov sploh ne predstavljamo več, saj ne bi dohajali povpraševanja in kakovosti, ki jo lahko dosegamo s stroji. V avtomobilistični industriji je v zadnjih letih največji poudarek ravno na avtomatizaciji. Veliko avtomobilov srednjega in višjega cenovnega razreda že ponuja avtomatsko bočno parkiranje, zaklepanje vrat, prižiganje luči, ko se stemni, prilagoditev luči glede na nasproti vozeče vozilo, sledenje črti, zaznavanje in upoštevanje prometnih znakov in še bi lahko naštevali.

Tržišče ponuja ogromno rešitev tudi na področju avtomatizacije objektov. Nekatere so bolj prilagodljive in znajo upravljati in avtomatizirati več elementov hiše, druge manj. Tudi cene rešitev zelo variirajo. Najcenejši sistemi se pri nas začnejo pri nekaj tisoč evrih, medtem ko se zgornje meje ne da določiti, saj je odvisna od neomejenih želja uporabnika [18].

V diplomski nalogi se bomo ukvarjali s krmiljenjem pametne hiše. Ker pa sistem ni namenjen samo hiši, ampak ga je mogoče vgraditi v katerikoli objekt, bomo uporabljali bolj splošno besedno zvezo: krmiljenje objekta. Na tržišču obstajajo različna stikala in elementi, ki omogočajo povezavo in krmiljenje na daljavo preko različnih tehnologij. Vendar so taka stikala draga in za večino velja, da jih z drugimi sistemi ne moremo krmiliti. Druga možnost je, da izberemo katerega izmed komercialnih sistemov za avtomatizacijo, pri čemer nas bo začetna investicija stala vsaj nekaj tisoč evrov. Zato smo se odločili, da sistem izdelamo sami in pri tem ohranimo obstoječe elemente v objektu, katerim bomo dodali še različne senzorje.

Izdelati želimo torej sistem za krmiljenje, s katerim bo mogoče upravljati na daljavo in nadzirati različne senzorje v obstoječem objektu. Upravljati želimo luči in motorje senčil ter multimedijske naprave, ki so v objektu že nameščene. Namestili bomo tudi več senzorjev za merjenje temperature in vlage ter senzor razdalje za merjenje količine vode v zbiralniku. Sistem za upravljanje objekta mora biti sposoben tudi postopne nadgradnje in vpeljave v objekt.

V naslednjem poglavju sledi podrobnejši opis področja avtomatizacije stavb. V tretjem poglavju bomo predstavili zasnovo sistema, vključno z arhitekturo in uporabljenimi elementi. V četrtem poglavju bomo opisali programske rešitve za strežniško aplikacijo, programe krmilnikov in vmesnik človek-stroj. V zadnjem poglavju bomo podali sklepne ugotovitve.

Poglavje 2 Pregled področja

Avtomatizacija objektov se dandanes veliko uporablja v poslovnih objektih in v objektih, prilagojenih starejšim osebam ter osebam z omejenimi motoričnimi sposobnostmi. Venomer pa je cilj avtomatizacije varnost, udobje in dolgoročni prihranek denarja.

Področja avtomatizacije so 48[23] [24]:

- nadzor porabe energije,
- prezračevanje, hlajenje in ogrevanje prostorov,
- varovanje,
- senčenje,
- kontrola multimedijskih vsebin in
- osvetljevanje prostorov.

Varovanje je pomemben dejavnik, saj sem moramo v poslovnih in bivalnih prostorih počutiti varno. Pa vendar je varnost širok pojem, ki ga lahko razdelimo v dva sklopa:

- varovanje oseb in
- varovanje osebne lastnine.

Prvi sklop predstavljajo razni sistemi za klic v sili in video nadzor, zato je še posebej pomemben za starejše in ljudi s posebnimi potrebami. Video nazor spada tudi v sklop varovanja osebnega premoženja. Vanj lahko klasificiramo še različne alarmne sisteme, kot sta na primer alarmni sistem za zaznavanje nevarnih plinov in požara. Od nekoga, ki veliko potuje, lahko pričakujemo, da bo varnosti posvečal več pozornosti kot udobju. Dobra stran okrepljene varnosti objekta je tudi možnost za nižje zavarovalne premije, kar pa je v Sloveniji bolj izjema kot pravilo.

Prihranek denarja lahko dosežemo z ugašanjem naprav v prostorih, ko v njih ni ljudi, in s pametnim nadzorom energetskih virov. Porabo slednjih lahko zmanjšamo z avtomatskim odzivom na zunanje okolje. V sončnih zimskih dneh odpremo senčila in tako v prostor spustimo naravno svetlobo in toploto ter s tem prihranimo na kurjavi. Poleti žaluzije čez dan zapiramo, da se prostori ne bi preveč segreli. Uporabimo lahko tudi urnike in v tistem delu dneva, ko v prostoru ni nikogar, ozračja na primer ne segrevamo oziroma hladimo. V zaprtih prostorih je zelo pomembno tudi ozračje, saj svež in čist zrak pripomore k delovni učinkovitosti in dobremu počutju, zato lahko ob zaznavi slabega zraka samodejno poženemo prezračevanje.

Če želimo sisteme za avtomatizacijo hiše prodajati, sta dva izmed popularnejših standardov za avtomatizacijo, ki se danes uporabljata, standard LonWorks [31] in standard KNX [1]. Nekoliko podrobneje bomo opisali standard KNX, ki je bil iz razvit treh starejših evropskih standardov:

- EHS (European Home System Protocol),
- BatiBUS in
- EIB (European Installation Bus) poznan tudi kot Instabus.

Standard združuje večino proizvajalcev elektronskih komponent za kontrolo pametnih hiš. Vse komponente različnih proizvajalcev, ki imajo certifikat KNX, so med seboj popolnoma združljive. Standard je osredotočen na področje poslovnih objektov in hiš, velika težava pa je težka vgradnja sistema za avtomatizacijo v že obstoječ objekt. Potrebno je namreč zamenjati vso obstoječo napeljavo. Težavo sicer lahko premostimo z brezžično komunikacijo med elementi, vendar pa to poviša ceno investicije. Za upravljanje naprav v takih sistemih se praviloma uporabljajo programabilni logični krmilniki (angl. Programmable Logic Controller). To so krmilniki z visoko zanesljivostjo in se uporabljajo na vseh področjih avtomatizacije. Specializirani so za delo z digitalnimi in analognimi signali in so zelo odporni na zunanje motnje [25], [26].

Sisteme, ki podpirajo standard KNX, lahko imenujemo centralni sistemi za avtomatizacijo, saj preko enotnega sistema krmilimo vse porabnike v objektu. Obstajajo pa tudi sistemi, ki omogočajo avtomatizacijo posameznega elementa. Tak sistem je na primer Phillips Hue [2], ki je neprimerno cenejši od centralnega, njegova namestitev pa od uporabnika ne zahteva posebnega znanja. Toda programska oprema in protokoli, ki se uporabljajo za komunikacijo, so praviloma zaprtega tipa, kar pa onemogoča povezovanje takih naprav v centralni sistem za nadzor. Taki sistemi praviloma tudi hitro zastarajo in ne omogočajo združljivosti z drugimi

podobnimi sistemi. Vseeno pa je na tem področju avtomatizacije zelo živahno, o čemer priča dejstvo, da sta se v letošnjem letu pojavili dve popolnoma novi podjetji, ki želita sistemu Hue odvzeti delež trga. Cena žarnice za sistem Hue v Sloveniji je okrog 60 EUR [27], najcenejši vstopni paket s sistemom za upravljanje in tremi žarnicami pa okrog 200 EUR [28].

Obstaja pa tudi nekakšna mešanica obeh zgoraj omenjenih tipov sistemov. To so sistemi, ki omogočajo upravljanje naprav preko posebnih vgradnih tipk, mobilnih aplikacij in ostalih naprav. Taki sistemi ponujajo različne module, s katerimi lahko vzpostavimo sistem za krmiljenje objekta. Moduli med seboj komunicirajo ob pomoči praviloma zaprtih protokolov, kot sta na primer X10 [29], Controlled Comfort [30] in Z-Wave [33]. Komunikacija s takimi moduli večinoma teče po visokonapetostni napeljavi, ki je speljana v objektu. To je seveda dobro, če želimo tak sistem uporabiti v obstoječem objektu. Slaba lastnost pa je, da so moduli dragi, izbira med proizvajalci pa majhna.

Poglavje 3 Zasnova sistema

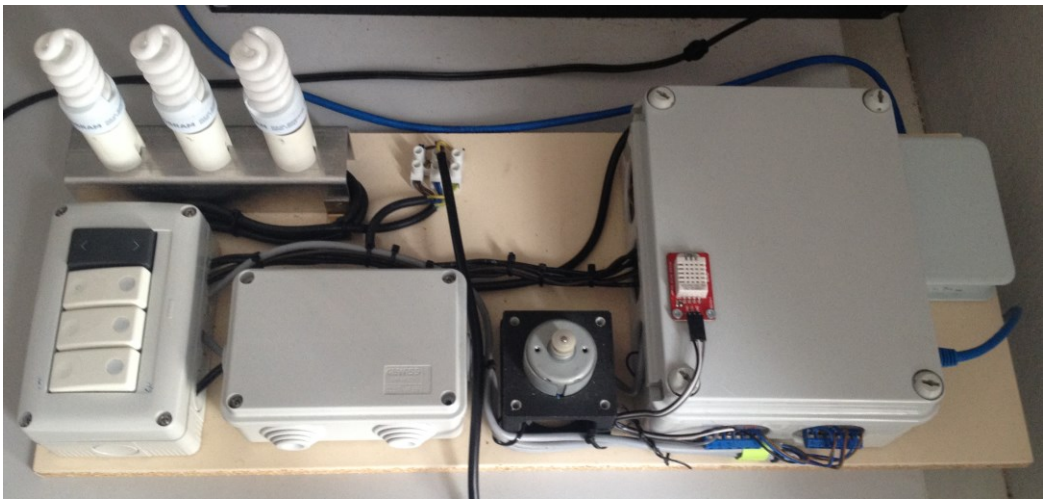
V pričujočem poglavju bomo predstavili arhitekturo sistema za krmiljenje objekta, naprave, ki ga sestavljajo, programsko opremo, ki smo jo uporabili za implementacijo sistema, ter glavne tehnologije, ki jih sistem uporablja.

Sistem, opisan v tej diplomski nalogi, je namenjen krmiljenju obstoječega objekta. Zato želimo izdelati sistem, ki bo zmožen krmiljenja več luči in enosmernih motorjev senčil ter bo cenovno ugoden in enostaven za vgradnjo.

Za potrebe razvoja sistema smo razvili prototip mladinske sobe, na kateri bomo izvedli test, s katerim bomo ocenili ceno investicije za celoten sistem. Prototip sobe je sestavljen iz treh različnih končnih porabnikov:

- tri luči,
- en motor in
- en temperaturni senzor.

Prototip sistema prikazuje slika 1. Med razvojem smo prototip razširili in nanj priključili še krmilnik z ultrazvočnim senzorjem.



Slika 1: Prototip sistema za krmiljenje objekta.

3.1 Arhitektura

Sistem je sestavljen iz štirih glavnih komponent:

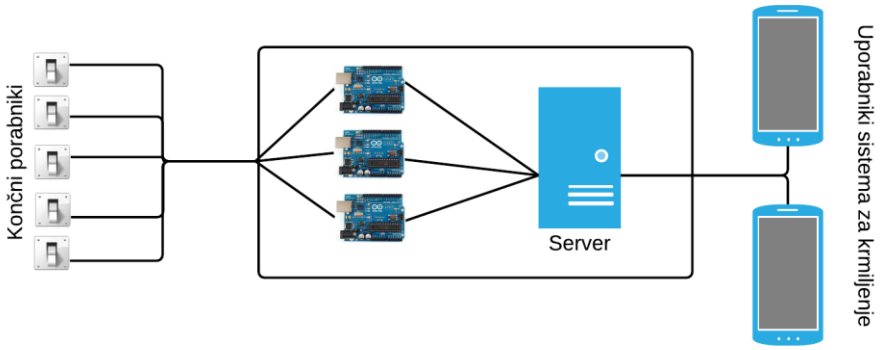
- končnih porabnikov,
- krmilnih naprav,
- strežnika in
- mobilne aplikacije za upravljanje s sistemom.

Za krmiljenje sistema smo uporabili krmilnike Arduino, predvsem zato, ker so enostavni za uporabo in imajo veliko uporabniško skupnost. Kot strežnik smo uporabili kartični računalnik Raspberry Pi 2, saj je na njem nameščen operacijski sistem, ki nam med drugim omogoča enostavno delo z več procesi, odpiranje množice hkratnih povezav z ostalimi napravami in namestitev podatkovne baze. Razlog za uporabo krmilnika Arduino in računalnika Raspberry Pi 2 je njuna popularnost; zanju je na voljo veliko literature tako v pisni kot tudi elektronski obliki.

Na en krmilnik lahko priključimo več končnih porabnikov, število teh pa je odvisno od njihovega tipa. Priključimo jih lahko največ šest, zato je krmilnikov v sistemu več. Na ta način dosežemo prostorsko neodvisnost sistema. Tako lahko v sistem vključimo tudi napravo, ki se ne nahaja na enaki geološki lokaciji.

Visokonapetostne porabnike, kot so na primer luči, krmilimo z releji, saj je izhodna napetost na krmilniku Arduino zgolj 5 V. Previsoka napetost prihaja tudi iz stikal, zato jo zmanjšamo ob pomoči vezja za zaznavanje fazne napetosti, katerega delovanje bomo predstavili v nadaljevanju. Izhod ploščice za zaznavanje fazne napetosti je povezan z razvojno ploščico Arduino.

Strežniško aplikacijo smo napisali v programskem jeziku Java in jo namestili na računalnik Raspberry Pi 2. Aplikacija predstavlja most oziroma storitev, ki zna na eni strani komunicirati s krmilniki naprav, na drugi pa z uporabnikom. Z njo zagotovimo, da uporabnik preko ene same vstopne točke nadzira vse končne porabnike v sistemu, kot prikazuje slika 2. Iz slike lahko razberemo tudi to, da uporabniku ni treba poznati nastavitev sistema in povezav med končnimi porabniki, krmilniki in strežnikom.



Slika 2: Prikaz strežnika kot mostu za komunikacijo med krmilniki in uporabniki. Komponente na sredinski škatli so končnemu uporabniku skrite.

Mobilna aplikacija je namenjena vsem napravam, na katerih teče operacijski sistem Android. V mislih smo imeli možnost razširitve aplikacije na mobilne naprave z drugimi operacijskimi sistemi, zato smo izbrali primerno razvojno okolje in tehnologije. Aplikacija je namenjena krmiljenju in spremljanju stanja končnih porabnikov v sistemu.

Stanje v objektu lahko krmilimo preko stikal ali sistema za krmiljenje. Ker je slednji močno povezan z napeljavo v objektu, moramo poskrbeti, da ob morebitni odpovedi sistema objekt še vedno lahko krmilimo ročno.

Kot krmilnik naprav v sistemu bi lahko uporabili kar računalnik Raspberry Pi 2, a se za to nismo odločili. Raspberry Pi 2 namreč za svoje delovanje potrebuje operacijski sistem, s katerim povečamo čas vzpostavitve sistema, če ta odpove. Krmilnik na drugi strani nima operacijskega sistema, saj na njem teče le program, ki smo ga napisali posebej zanj, zato je vzpostavitveni čas občutno krajši. Drugi razlog je, da so krmilniki primernejši za delo z različnimi senzorji, motorji in svetilnimi elementi. Imajo namreč močnejše izhode. Na krmilniku Arduino lahko na posamezni izhodni priključek spustimo tok do 40 mA, medtem ko na Raspberry Pi 2 največ 10 mA. To je za določene elemente premalo, zato za njihovo vezavo potrebujemo dodatna vezja. Raspberry Pi 2 podpira samo vhodno napetost 3,3 V na splošno namenskih priključkih, medtem ko na krmilniku Arduino izbiramo med napetostjo 3,3 V in 5 V. V sistemu uporabljamo 5 V napetosti na vseh elementih, priključenih neposredno na krmilnik. To nam zaradi tesne povezanosti med napeljavo in sistemom za krmiljenje bolj ustreza, saj pri nižjih napetostih lažje pride do motenj pri preklapljanju končnih porabnikov. Dodaten razlog za izbiro krmilnika Arduino je občutno manjša poraba energije. Raspberry Pi 2 potrebuje za delovanje 700 mW, razvojna ploščica Arduino pa le 175 mW.

3.2 Naprave

V tem podpoglavju bomo predstavili naprave in nekatere pomembne elemente, ki smo jih uporabili pri izdelavi sistema.

3.2.1 Razvojna ploščica Arduino Uno

Arduino Uno je razvojna ploščica, zasnovana na mikrokrmilniku ATmega328P s frekvenco delovanja 16 MHz. Slika 3 prikazuje razvojno ploščico Arduino Uno. Na razpolago ima 14 digitalnih in 6 analognih splošno namenskih priključkov, ki jih lahko uporabljamo kot vhodne ali izhodne priključke. Na voljo imamo tudi 32 KB bliskovnega pomnilnika za shranjevanje programa, od katerega moramo 0,5 KB takoj prepustiti zagonskemu nalagalniku, 2 KB pomnilnika SRAM moramo prihraniti za spremenljivke programa, 1 KB pomnilnika EEPROM pa za trajno shranjevanje podatkov. Možnosti za napajanje sta dve, preko vrat USB ali zunanjega 5 V napajanja.

Slika 3: Razvojna ploščica Arduino Uno.

Vsak krmilnik Arduino v sistemu pred vgradnjo ustrezno sprogramiramo za upravljanje z napravami, ki so nanj priključene.

3.2.1.1 Razvojna ploščica Arduino Ethernet in razširitvena ploščica Arduino Ethernet Shield

Razvojna ploščica Arduino Ethernet je zgrajena na enaki osnovi kot razvojna ploščica Arduino Uno, zato bomo tu opisali samo najpomembnejše razlike med njima. Na voljo imamo vrata Ethernet, ki so povezana z Ethernet krmilnikom Wiznet5100. Slednji podpira do štiri hkratne odprte povezave. Ker ploščica nima vrat USB, jo moramo programirati preko

povezave USB TTL (Transistor-Transistor logic). Slika 4 prikazuje kabel za programiranje razvojne ploščice Arduino Ethernet. Dodan ima tudi vhod za kartico mikro SD.

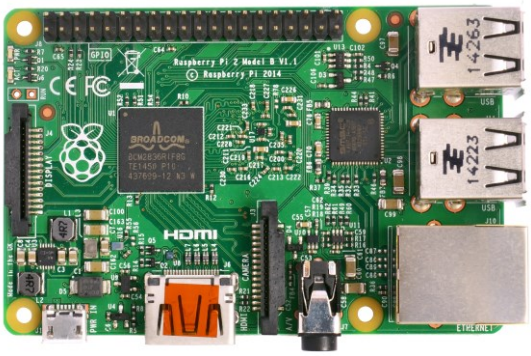
Slika 4: Kabel USB TTL-232R-5V za programiranje razvojne ploščice Arduino Ethernet.

V sistemu smo poleg razvojne ploščice Arduino Ethernet uporabili tudi razširitveno ploščico Ethernet Shield. Ta je združljiva z večino razvojnih ploščic iz družine Arduino in ima enake lastnosti kot razvojna ploščica Arduino Ethernet, le da ne vsebuje mikrokrmilnika ATmega, ampak z njim komunicira preko vmesnika SPI (Serial Peripheral Interface Bus).

3.2.2 Raspbery Pi 2

Za strežnik smo izbrali kartični računalnik Raspbery Pi 2, ki uporablja 4-jedrni procesor ARM Cortex-A7 s frekvenco delovanja 900 MHz ter 1 GB pomnilnika RAM. Predstavljen je na sliki 5. Uporabili smo odprtokodni operacijski sistem Raspbian, ki je enostaven za namestitev in uporabo. Osnovan je na Linux distribuciji Debian.

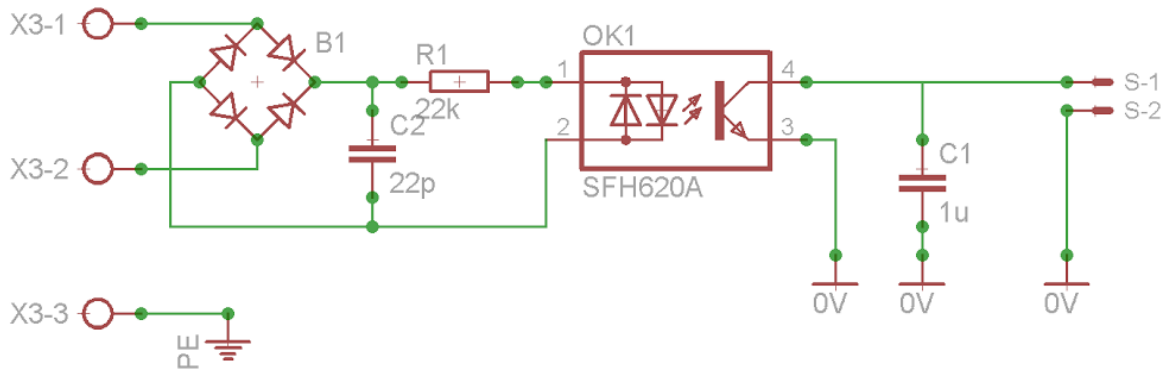
Računalnik ima med drugim 40 splošno namenskih priključkov, vrata HDMI, vhod za kartico mikro SD, vrata Ethernet in vhod avdio.



Slika 5: Kartični računalnik Raspberry Pi 2

3.2.3 Ploščica za zaznavanje fazne napetosti

Namen ploščice je zaznavanje fazne napetosti na visokonapetostnih porabnikih v sistemu. Ker jih ne moremo priključiti neposredno na krmilnik, smo razvili ploščico, ki nam pomaga spremljati stanje na visokonapetostnih porabnikih. Na sliki 6 je prikazana shema ploščice. Na vhod ploščice pripeljemo fazo, ničlo ter ozemljitev iz stikala končnega porabnika. Vhodi so označeni z X3. Če je prisotna napetost na porabniku, skozi diodni mostič (Greatzov mostič) polnimo kondenzator C2. Nato z uporom zmanjšamo tok, ki priteče v optični sklopnik. Slednji galvansko loči vhode krmilnika Arduino od visokonapetostnih porabnikov in ob prisotnosti napetosti poveže vsebujoči foto-tranzistor na izhod ploščice S1. Izhod ploščice je povezan na enega izmed vhodov v krmilnik končnih naprav preko upora na krmilniku. S tem zagotovimo, da je vrednost na vhodu krmilnika definirana, kadar je stikalo izključeno, in se tako izognemo morebitnim motnjam.



Slika 6: Shematski prikaz ploščice za zaznavanje fazne napetosti.

3.2.4 Releji

V sistemu uporabljamo releje za krmiljenje visokonapetostnih končnih porabnikov. Pri realizaciji sistema smo uporabili ploščico YL-53 z osmimi releji. Kako ploščica zgleda prikazuje slika 7. To so stikala, ki imajo krmilni tokokrog galvansko ločen od delovnih kontaktov. Uporabljamo jih, ko je potrebno vezje oziroma v našem primeru napeljavo krmiliti z nizkimi napetostnimi signali. Končni porabniki v sistemu potrebujejo za svoje delovanje večje napetosti in tokove, kot jih premoremo na krmilniku Arduino. Zato releje krmilimo z nizkimi napetostmi, s katerimi določamo stanje stikal, na katera pripeljemo želeno napetost za končnega porabnika.

Pomembne lastnosti relejev so:

- maksimalna napetost med razklenjenimi kontakti,
- maksimalen tok preko sklenjenih kontaktov,
- maksimalna frekvenca preklopov.

Releji, uporabljeni na ploščici prenesejo maksimalen tok 10 A in maksimalno napetost 250 V. To pomeni, da lahko z njimi upravljamo končne porabniki do moči 2,5 kW. Z razvojno ploščico Arduino lahko krmilimo porabnike katerih maksimalna moč je zgolj 2 mW.

Podroben opis krmiljenja z releji je opisan v poglavjih 4.2.2 in 4.2.3.

Slika 7: Ploščica YL-53 z osmimi releji.

3.2.5 Senzorji

3.2.5.1 Ultrazvočni senzor

Ultrazvočni senzor, prikazuje ga slika 8, potrebujemo za merjenje količine vode v zbiralniku. Lahko bi ga uporabili tudi za avtomatizacijo prižiganja luči na vhodu v objekt. V našem primeru smo uporabili senzor HC-SR04, saj je cenovno ugoden in za naše potrebe dovolj zmogljiv. Njegov domet po specifikacijah znaša od 0,02 m do 4 m z natančnostjo do 3 mm.

Slika 8: Ultrazvočni senzor HS-SR04.

Ultrazvočni senzor aktiviramo s pulzom, ki sproži ciklično nihanje signala na frekvenci 40 kHz. Zvočni signal se od objekta, v katerega je senzor usmerjen, odbije. Na podlagi časa, ki preteče od pošiljanja zvoka do prejema odmeva, izračunamo razdaljo med objektom in senzorjem. Slabost ultrazvočnih senzorjev je začetno mrtvo območje in širok zvočni stožec. Z večanjem razdalje se energija v odmevu zmanjšuje, kar je posledično razlog za omejitev dometa senzorja.

Senzor ni primeren za krmiljenje z računalnikom Raspberry Pi 2, saj za delovanje potrebuje 5 V izhodne napetosti in 20 mA toka.

3.2.5.2 Senzor temperature in vlage

Za merjenje temperature v prostoru smo uporabili senzor DHT22, ki ga prikazuje slika 9. Zmožen je merjenja temperature na območju od -40 do 80 °C in vlage na območju od 0 do 100 %. Točnost temperature lahko izmerimo z natančnostjo 0,5 °C, vlažnost pa z natančnostjo 5 %.

Slika 9: Senzor za merjenje temperature in vlage DHT22.

Senzor za svoje delovanje potrebuje od 3,3 V do 5 V enosmerne napajalne napetosti. Poleg dveh priključkov, na katera priključimo napajalno napetost in ozemljitev, imamo še en priključek za komunikacijo. Krmilnik s senzorjem komunicira s poenostavljeno različico komunikacije 1-Wire Bus, ki uporablja model gospodar-suženj. Vlogo sužnja ima senzor, ki odgovarja na zahteve gospodarja – v našem primeru je to krmilnik.

3.3 Tehnologije

Vse naprave v sistemu med seboj komunicirajo preko protokola TCP/IP. Zato ima vsaka naprava svoj naslov IP, ki ji ga dodeli usmerjevalnik na podlagi naslova MAC.

Za izvedbo sistema smo uporabili več programskih jezikov. Krmilnike Arduino smo programirali v programskem jeziku C++ in prevajali s prevajalnikom avr-gcc, ki je del razvojnega okolja Arduino Studio IDE. Strežniško aplikacijo smo programirali v Javi predvsem zaradi prenosljivosti med operacijskimi sistemi in enostavnosti jezika za pisanje večnitnih programov. Mobilno aplikacijo za operacijski sistem Android smo napisali v jeziku C#. Za ta neobičajen pristop smo se odločili zaradi možnosti razširitve aplikacije na operacijski sistem iOS. Da bi imeli pri tem karseda malo težav, smo aplikacijo izdelali ob pomoči platforme Xamarin (pred tem MONO), ki omogoča pisanje platformno neodvisnih mobilnih aplikacij v programskem jeziku C#.

3.3.1 Xamarin

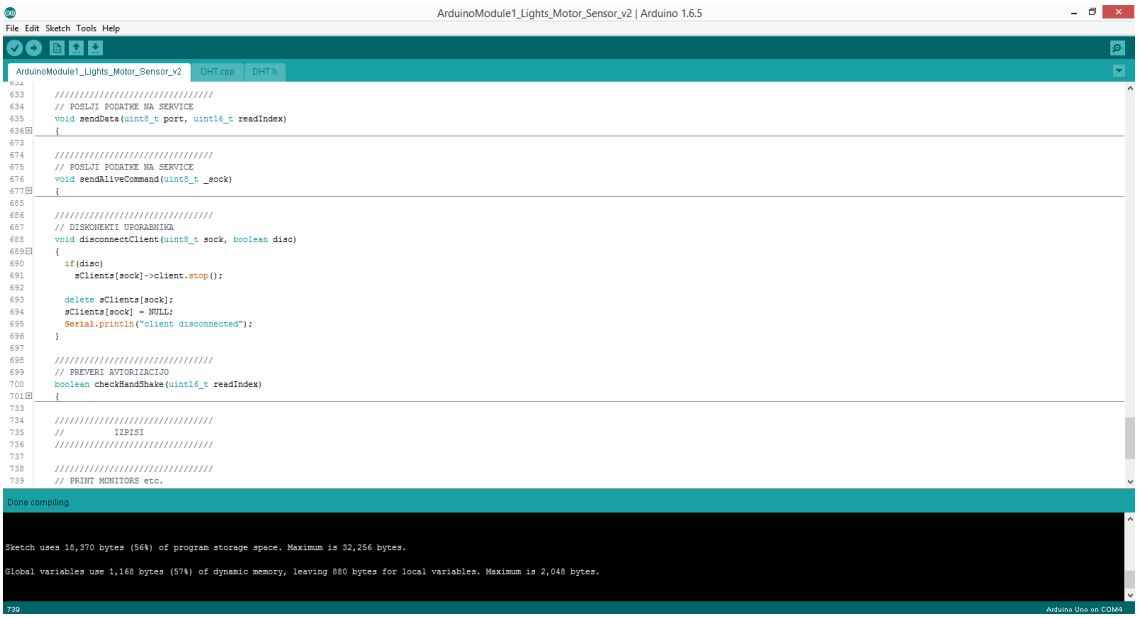
Xamarin je plačljiva platforma za razvoj aplikacij, namenjenih za različne mobilne operacijske sisteme. Študenti lahko zaprosimo za enoletno študentsko licenco, ki se od poslovne licence razlikuje predvsem v tem, da razvitih aplikacij ne smemo prodajati.

Če želimo doseči kompatibilnost mobilne aplikacije na različnih mobilnih operacijskih sistemih, moramo uporabiti tehnologijo Xamarin.Forms. Vseeno smo občasno primorani določene operacije implementirati za vsak operacijski sistem posebej. Z upoštevanjem pravil strukturiranja programske kode lahko poskrbimo, da vedno kličemo enako metodo, ki pa se bo izvedla za ustrezni operacijski sistem.

3.4 Programska oprema, uporabljena pri razvoju sistema

Med razvojem sistema smo uporabili več programskih okolij. Za programiranje krmilnikov Arduino smo uporabljali Arduino Studio v1.6.5. Prikazuje ga slika 10. Razvojno okolje je prosto dostopno na spletni strani podjetja Arduino. Programe za krmilnike Arduino smo pisali

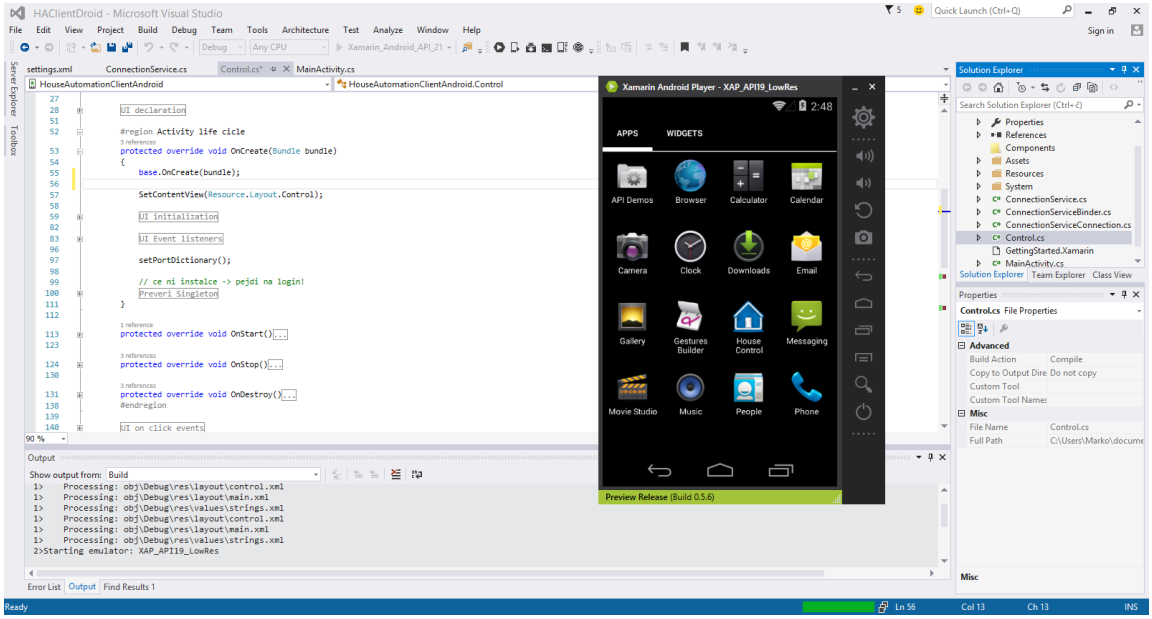
v programskem jeziku C++. Razvojno okolje nam ponuja vse potrebne funkcije, kot so: monitor za serijsko komunikacijo, prevajalnik, programator in prednastavitve za večino ploščic iz družine Arduino, ki jih potrebujemo za uspešno nalaganje programa.



Slika 10: Arduino Studio v1.6.5.

Pri programiranju strežniške aplikacije smo uporabljali razvojno okolje NetBeans. Slednje je alternativa priljubljenemu razvojnemu okolju Eclipse IDE, a podpira tudi gradnjo grafičnega vmesnika. V razvojnem okolju NetBeans je dobro podprto tudi razhroščevanje programske kode. Okolje je prosto dostopno in odprtokodno, sicer pa za razvoj skrbi podjetje Oracle.

Mobilno aplikacijo smo programirali v razvojnem okolju Microsoft Visual Studio 2015, ki ga prikazuje slika 11. Nekateri paketi programske rešitve so brezplačni, nekateri napredni paketi z več funkcionalnostmi pa plačljivi. V okolje smo vključili tudi platformo Xamarin, ki med drugim vključuje razhroščevanje programske opreme, gnane na virtualnem stroju z operacijskim sistemom Android in grafični vmesnik za gradnjo grafičnih pogledov. Ker je osnovni virtualni stroj, ki ga dobimo v paketu Android SDK (Software Development Kit), zelo neodziven in počasen, smo ga zamenjali z optimiziranim virtualnim strojem Xamarin Android Player, ki je zgrajen na programu VirtualBox podjetja Oracle. Izbiramo lahko med operacijskimi sistemi Android različic 4.1.1, 4.2.2 in 5.1.0 ter napravami Nexus 4, 5, 7 in 10. V diplomski nalogi smo uporabili operacijski sistem Android 4.2.2 in ga gnali na virtualnem stroju, ki simulira napravo Nexus 4.



Slika 11: V ozadju Microsoft Visual Studio 2015, spredaj Xamarin Android Player.

Poglavje 4 Opis sistema

Namen sistema za krmiljenja objekta je, da lahko uporabnik v vsakem trenutku preveri njegovo trenutno stanje, in ga po potrebi spremeni. Stanje lahko opazuje z mobilno aplikacijo, ki se poveže s sistemom preko svetovnega spleta. Stanje objekta torej lahko opazujemo kjerkoli se lahko povežemo na splet. Sistem omogoča nadzor in pregled stanja motorjev in luči ter pregled stanja temperaturnih senzorjev in senzorja razdalje, ki se uporablja za merjenje količine vode v zbiralniku.

Mobilna aplikacija komunicira s strežniško aplikacijo, ki skrbi za spremljanje stanj na vseh krmilnikih v sistemu. S tem poskrbimo, da imamo na enem mestu shranjena vsa stanja končnih porabnikov v sistemu. Strežniška aplikacija je zadolžena za vzpostavitev in ohranjanje povezave s krmilniki. Za varnost v sistemu poskrbimo s prijavnimi postopki. Uspešno prijavo zahteva krmilnik, preden odjemalcu, v našem primeru strežniški aplikaciji, dovoli krmiljenje sistema. Strežniška aplikacija zahteva tudi uspešno prijavo z uporabniškim imenom in geslom. Shranjena sta v podatkovni bazi na strežniku.

Prijavljanje v krmilnik je s strani strežniške aplikacije avtomatizirano, prijava z mobilno aplikacijo na strežnik pa ne. Zaradi varnosti mora uporabnik ob vsaki prijavi v sistem vpisati uporabniško ime in geslo. Postopke prijave bomo v nadaljevanju podrobneje predstavili.

4.1 Komunikacija

Vsi elementi v sistemu za krmiljenje objekta omogočajo komunikacijo preko protokola TCP/IP. Med seboj si pošiljajo nize podatkov, ki predstavljajo stavek elementu, ki mu je namenjen. Mobilna aplikacija in strežnik tvorita ukazne stavke, s katerimi krmilnike povprašujemo po trenutnem stanju ali zahtevamo njegovo spremembo. Krmilniki odgovarjajo s statusnimi stavki, s katerimi vrnejo trenutno stanje sistema. Krmilnik sporoča svoje stanje, ko se to spremeni. Enake statusne stavke pošilja tudi strežnik mobilni aplikaciji, ko ji sporoča stanje v sistemu.

Ukazni stavek je sestavljen, kot prikazuje tabela 1. Polje modul predstavlja identifikacijsko številko krmilnika ali strežnika, ki mu je ukaz namenjen. S poljem vrata naslavljamo končnega porabnika na krmilniku, ki ga želimo krmiliti ali ugotoviti njegovo trenutno stanje.

Polje tip nam določa, ali gre za ukaz ali povpraševanje. S poljem podatki pa določamo novo stanje končnega porabnika v sistemu, ko gre za ukaz spremembe stanja. V posebnih primerih, kot je prijava, v polju podatki pošiljamo uporabniška imena in gesla za prijavo. Če izvajamo povpraševanje, je polje podatki nepomembno in zavzame vrednost nič.

	Modul	**Vrata**	**Tip**	**Podatki**
Velikost	1 bajt	1 bajt	1 bajt	1 bajt ali več

Tabela 1: Zgradba ukaznega stavka.

Krmilnik nam odgovori s statusnim stavkom. Sestavljen je, kot prikazuje tabela 2. Polje modul nam pove, kateremu krmilniku pripada status, polje vrata nam opisujejo, o katerem končnem porabniku sledijo podatki. Nato sledi polje podatki z enim ali več bajtov podatkov.

	Modul	**Vrata**	**Podatki**
Velikost	1 bajt	1 bajt	1 bajt ali več

Tabela 2: Zgradba statusnega stavka.

Vsak krmilnik ima svojo identifikacijsko številko, poimenovano modul, katere vrednost je v sistemu unikatna in večja od nič. Identifikacijska številka nič predstavlja strežniško aplikacijo v sistemu.

Med strežnikom in krmilnikom poznamo dva posebna ukazna stavka: za prijavo in preverjanja statusa povezave. Prvi se uporabi za prijavo v krmilnik in je zasnovan kot ostali ukazni stavki z razliko polja podatki – to obsega pet bajtov in predstavlja geslo krmilnika. Primer stavka prikazuje tabela 3. Da gre za prijavo v krmilnik, napovemo z vrednostjo 0xFF polj vrata in tip. Krmilnik sporoči uspešnost prijave s statusnim stavkom, katerega vrednost polja podatki je 0xAC, če je bila prijava uspešna, sicer je vrednost polja podatki 0xAF. Primer statusnega stavka ob uspešni prijavi prikazuje tabela 4. Ukazni stavek za preverjanje statusa povezave ima vrednost polja vrata enako kot prijavni stavek, polje tip pa zavzame vrednost 0xA0. Primer ukaznega stavka za preverjanje povezave prikazuje tabela 5. Vrednost polja podatki je nepomembna in lahko zavzame katerokoli vrednost. Krmilnik na prejeto sporočilo strežniku ne odgovori.

	Modul	**Vrata**	**Tip**	**Podatki**
Vrednost	0x02	0xFF	0xFF	0x0102030405

Tabela 3: Primer prijavnega ukaznega stavka za krmilnik.

	Modul	**Vrata**	**Tip**
Vrednost	0x02	0xFF	0xAC

Tabela 4: Primer statusnega stavka ob uspešni prijavi.

	Modul	**Vrata**	**Tip**	**Podatki**
Vrednost	0x02	0xFF	0xA0	0x00

Tabela 5: Primer ukaznega stavka za ohranjanje vzpostavljene povezave.

Podoben prijavni stavek, kot ga poznamo med strežnikom in krmilnikom, imamo tudi za prijavo mobilne aplikacije na strežnik. Sestavljen je enako kot ukazni stavek za prijavo v krmilnik, le v polju podatki pošiljamo uporabniško ime in geslo za prijavo v strežniško aplikacijo. Podatki so dolgi 70 B, prvih 30 B predstavlja uporabniško ime, ostalih 40 B pa geslo, kodirano s kriptografskim algoritmom SHA1. Strežniška aplikacija odgovarja mobilni aplikaciji z enakimi statusnimi stavki, kot jih prejema ob prijavi v krmilnik.

4.2 Krmilniki naprav

V sistemu za krmiljenje objekta je več krmilnikov, ki lahko opravljajo različne naloge. Za to poskrbimo s programiranjem vsakega krmilnika posebej. Ker smo uporabljali krmilnike Arduino Uno z razširitveno ploščico Arduino Ethernet Shild in krmilnike Arduino Ethernet, imamo pri vsakem na voljo šest analognih in štirinajst splošno namenskih vhodno-izhodnih priključkov. Tip priključka določimo v nastavitveni rutini programa vsakega krmilnika. Primer nastavitev priključkov prikazuje slika 12.

```
////////////////////////////////
// SETUP PORT TYPES
// INPUTS
pinMode(L1_IN, INPUT_PULLUP);        // Light 1 physical status
pinMode(L2_IN, INPUT_PULLUP);        // Light 2 physical status
pinMode(L3_IN, INPUT_PULLUP);        // Light 3 physical status
pinMode(M1_U, INPUT_PULLUP);         // Motor GOR signal, to bo D2
pinMode(M1_D, INPUT_PULLUP);         // Motor DOL signal, to bo D3

////////////////////////////////
// SETUP DHT
dht.setup(DHT_IN);
minSamplePeriod = dht.getMinimumSamplingPeriod();

////////////////////////////////
// OUTPUTS = Relay control
pinMode(L1_C, OUTPUT);               // Light 1, control
pinMode(L1_S, OUTPUT);               // Light 1, status
pinMode(L2_C, OUTPUT);
pinMode(L2_S, OUTPUT);
pinMode(L3_C, OUTPUT);
pinMode(L3_S, OUTPUT);
pinMode(M1_1, OUTPUT);               // Motor 1, Up
pinMode(M1_2, OUTPUT);               // Motor 1, Down
```

Slika 12: Nastavitve priključkov na enem izmed krmilnikov v prototipu sistema za krmiljenje objekta.

Vsak krmilnik ima svojo identifikacijsko številko, imenovano modul, geslo za prijavo v krmilnik dolžine pet bajtov in časovni konstanti ACK_TIME in PASS_TIME. Za pridobitev internetnega naslova potrebujemo veljaven naslov MAC, ki ga vpišemo v program krmilnika. Nekateri uporabljeni krmilniki naslova MAC niso imeli označenega na ploščici, zato smo si ga morali izmisliti. Naslov je statično določen s prvimi petimi bajti naslova enega od krmilnikov, zadnji bajt pa določa identifikacijska številka krmilnika. Tako lahko pri nastavljanju usmerjevalnika zlahka nastavimo statičen naslov IP za vsak krmilnik, saj poznamo njegov naslov MAC. Slika 13 prikazuje nastavitev naslova MAC ter ostalih spremenljivk za komunikacijo.

```
////////////////////////////////
// ARDUINO COMM SETTINGS
const int BAUD_RATE = 9600;
const uint16_t PORT = 5000;
byte MAC[] = {0x90, 0xA2, 0xDA, 0x0F, 0x1A, MODULE};
IPAddress ip(192, 168, 1, 101);
IPAddress subnet(255, 255, 255, 0);
IPAddress gateway(192, 168, 1, 1);
EthernetServer server(PORT);
EthernetClient client;
```

Slika 13: Nastavitve komunikacijskih spremenljivk enega izmed krmilnikov v prototipu sistema za krmiljenje objekta.

V konstanti ACK_TIME je shranjen čas v milisekundah, ki predstavlja maksimalni časovni okvir med dvema prejetima ukaznima stavkoma. Če ACK_TIME prekoračimo, krmilnik zapre povezavo. Konstanta PASS_TIME predstavlja čas v milisekundah, v katerem mora strežnik poslati geslo za prijavo v krmilnik. Čas se začne meriti, ko na krmilniku zaznamo vzpostavljeno povezavo. Če v tem času krmilnik ne prejme gesla, se povezava zapre. Povezava se zapre tudi v primeru napačno vnesenega gesla za prijavo v krmilnik. Povezave smo primorani zapirati, ker imamo na voljo le štiri aktivne povezave hkrati. V primeru večkratnega izpada omrežja bi se bazen povezav hitro zapolnil in potreben bi bil ponoven zagon krmilnika.

Podatke o vsakem povezanem odjemalcu, kot na primer: naslov IP, programsko vtičnico povezave, čas zadnjega prejetega ukaznega stavka in status prijave v krmilnik hranimo v objektu ServiceClient.

Krmilimo in spremljamo lahko stanja luči in motorjev ter spremljamo stanja senzorjev za temperaturo in vlažnost ter senzorjev razdalje. Glavna rutina, ki jo opravlja krmilnik, je sestavljena iz dveh sklopov. V prvem preverimo, ali se je spremenilo stanje na stikalih, v drugem sklopu pa, ali smo prejeli ukazni stavek, in ga obdelamo. V primeru spremembe stanja sistema odjemalce o tem obvestimo.

V nadaljevanju bomo opisali, kako deluje krmiljenje in spremljanje stanj v določenih primerih.

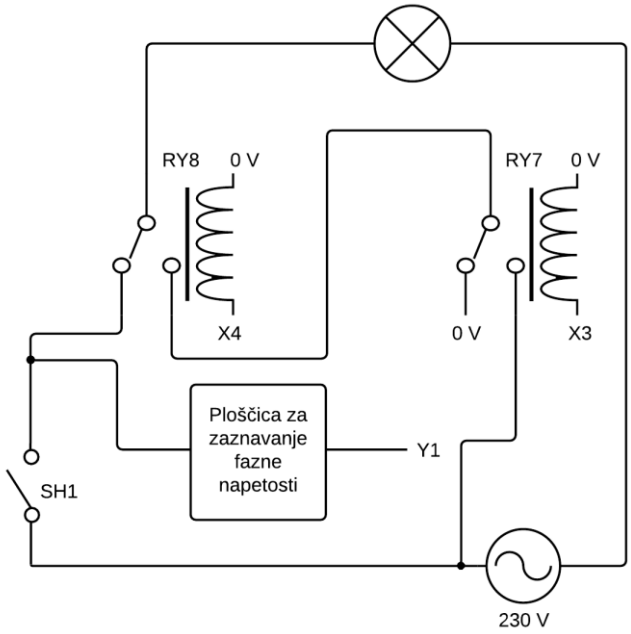
4.2.1 Prijava v krmilnik

Pri povezovanju strežnika s krmilnikom smo naleteli na težave, ker sprva nismo mogli zaznati povezane strežniške aplikacije. Obstoječa knjižnica, ki jo ponuja podjetje Arduino, zazna aktivnega odjemalca le v primeru, ko nam ta pošlje podatke. Ker pa imamo po specifikacijah

krmilnika Wiznet5100, ki skrbi za povezave Ethernet, možnost za detekcijo povezanega odjemalca [3], tudi če ta ne pošlje podatkov, smo bili primorani knjižnico razširiti. To smo storili z metodo, ki nam vrača stanje programske vtičnice, katere identifikacijsko številko pošljemo kot argument metode. Vsaka programska vtičnica predstavlja eno povezavo z odjemalcem. Če je v povezanem stanju, pomeni, da se je nekdo povezal na krmilnik in ima v določenem času možnost poslati geslo. Ta čas je določen s konstanto PASS_TIME. Če tega ne naredi, krmilnik zapre povezavo in programsko vtičnico postavi v zaprto stanje. Enako se zgodi tudi v primeru, ko odjemalec pošlje napačno geslo, vendar se v tem primeru odjemalca obvesti s statusnim stavkom. Ko je odjemalec, v našem primeru strežniška aplikacija, prijavljen v krmilnik, je ta od njega pripravljen sprejeti ukazne stavke, s katerimi lahko spremenimo stanje v sistemu.

4.2.2 Krmiljenje luči

Za vsako luč, ki jo krmilimo, uporabimo en vhod in dva izhoda na krmilniku Arduino. Vhod predstavlja stanje na stikalu, izhoda pa dva releja, s katerima krmilimo luč.



Slika 14: Shematski prikaz vezave luči v sitemu za krmiljenje objekta.

Slika 14 prikazuje vezavo luči v sistemu za krmiljenje. Releja RY8 in RY7 sta priključena na izhod krmilnika Arduino. Rele RY8 določa, ali luč krmilimo s stikalom ali z izhodom krmilnika X3 in ga imenujemo kontrolni rele. Njegovo stanje krmilimo z izhodom krmilnika X4. Rele RY7 imenujemo statusni rele. Z njim določamo stanje luči, ko je stanje na izhodu krmilnika X4 aktivno. Statusni rele krmilimo z izhodom krmilnika X3. Stikalo je s

krmilnikom povezano preko ploščice za zaznavanje fazne napetosti. Status stikala, ki ga preberemo na krmilniku, predstavlja vhod Y1. Krmilnik ob vsakem obhodu glavnega programa bere vrednost vhoda Y1 in ga primerja s prebrano vrednostjo ob prejšnjem obhodu programa. Če se vrednosti razlikujeta, je prišlo do spremembe stanja na stikalu. Ko se to zgodi, krmilnik na izhod X4 postavi neaktivno stanje in s tem pa povzroči, da se stanje luči določa glede na trenutno stanje stikala. Ob prejetem ukaznem stavku se na izhod krmilnika postavi aktivno stanje in s tem povzroči, da status luči določamo s statusnim krmilnikom RY7. Če je na izhodu X3 aktivno stanje, bo luč svetila, v nasprotnem primeru pa bo ugasnjena. Stanje na izhodu X3 določamo na podlagi polja podatki v prejetem ukaznem stavku. Če je vrednost polja podatki 0x00, luč ugasnemo, vrednost 0x01 pomeni prižig luči. Ob vsaki spremembi stanja sistema krmilnik s statusnim stavkom obvesti vse odjemalce o novem stanju.

Ko na krmilniku spremenimo stanje, se to na stikalu ne pozna, saj ga s krmilnikom ne moremo fizično preklapljati. Denimo, da uporabnik želi fizično prižgati luč in je stikalo že sklenjeno, saj smo luč ugasnili s krmilnikom. Uporabnik mora v tem primeru stikalo najprej razkleniti in nato ponovno skleniti. Krmilnik ob prvi zaznavi spremembe stanja na stikalu preklopi kontrolni rele v pozicijo, da stanje luči narekuje stikalo, toda to je razklenjeno in luč je še vedno ugasnjena. Zato mora uporabnik, če želi prižgati luč, stikalo skleniti.

4.2.3 Krmiljenje motorja

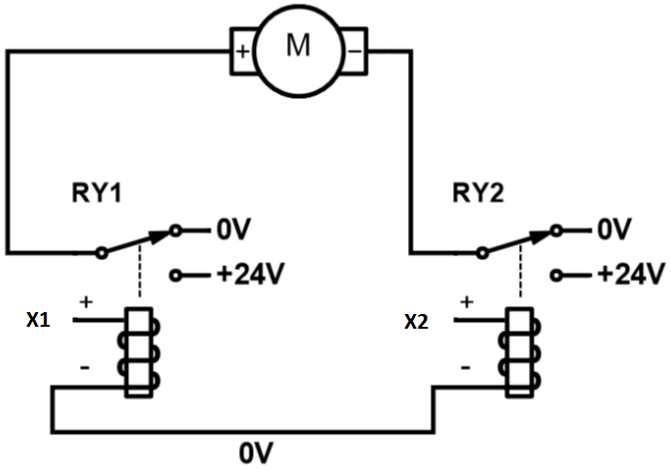
Vsak motor, priključen na krmilnik, ima po dva vhoda in izhoda. Vhoda nam predstavljata stanje stikala za pomik motorja v smer urinega kazalca ali v nasprotno smer. Izhoda predstavljata dva releja, s katerima krmilimo motor. Ko krmilnik prejme ukazni stavek za pomik motorja, požene motor v želeno smer. Smer motorja se v ukaznem stavku določi z vrednostjo polja vrata. Tabela 6 prikazuje primer ukaznega stavka za pomik motorja.

	Modul	**Vrata**	**Tip**	**Podatki**
Vrednost	0x01	0x02	0x02	0x01

Tabela 6: Primer ukaznega stavka za premik motorja v smeri urinega kazalca.

Motor je v pogonu določen čas, katerega določa konstanta MOTOR_TIME in predstavlja izmerjen čas za pomik žaluzij od vrha do dna okna. Če v času, ko se motor premika, krmilnik prejme še en ukaz za pomik motorja, se ta ustavi. Na tak način ga lahko poljubno krmilimo preko mobilne aplikacije. Če ga krmilimo ročno, lahko izbiramo med avtomatskim in polavtomatskim načinom. Avtomatski način predstavlja pritisk na stikalo, ki traja manj kot eno sekundo. V tem primeru se motor pomika določen čas v izbrano smer. Pomikanje motorja

v avtomatskem načinu lahko prekinemo s pritiskom na stikalo, medtem ko se motor vrti. S pritiskom, daljšim od ene sekunde, krmilniku povemo, da bomo motor krmilili v polavtomatskem načinu. Krmilnik začne vrteti motor v izbrano smer in se vrti, dokler stikala ne spustimo. Ob spremembi stanja krmilnik vsem odjemalcem pošlje statusni stavek z informacijo, v katero smer se je motor nazadnje premikal.



Slika 15: Shematska vezava enosmernega motorja za krmiljenje senčil.

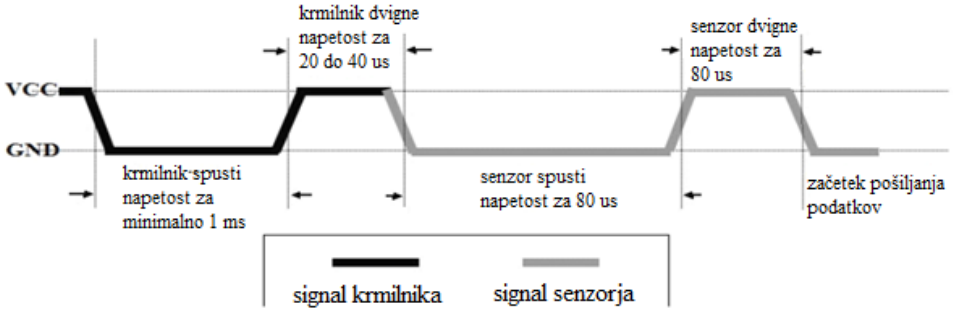
Slika 15 prikazuje vezavo motorja v sistem krmiljenja hiše, pri čemer X1 in X2 predstavljata izhoda iz krmilnika. Če je na izhodu krmilnika, označenim z X1, aktivno stanje, je rele RY1 aktiven in motor vrtimo v smeri urinega kazalca. Enako velja za rele RY2 v povezavi z izhodom krmilnika X2, vendar se v tem primeru motor pomika v nasprotni smeri urinega kazalca. Če je na izhodih X1 in X2 aktivno stanje, sta oba releja, RY1 in RY2, aktivna, in se motor ne premika.

4.2.4 Senzor temperature in vlage

Senzor DHT22 komunicira s krmilnikom preko enega vhoda z uporabo poenostavljene različice protokola 1-Wire Bus. Uporabili smo knjižnico arduino-DHT [4], iz katere smo izbrisali nepotrebne funkcije za podporo ostalih senzorjev. Senzor podpira branje stanja vsaki dve sekundi, zato s tako frekvenco tudi vzorčimo. Ko prepoznamo spremembo temperature ali vlage, se novo stanje pošlje vsem odjemalcem, ki so povezani na krmilnik.

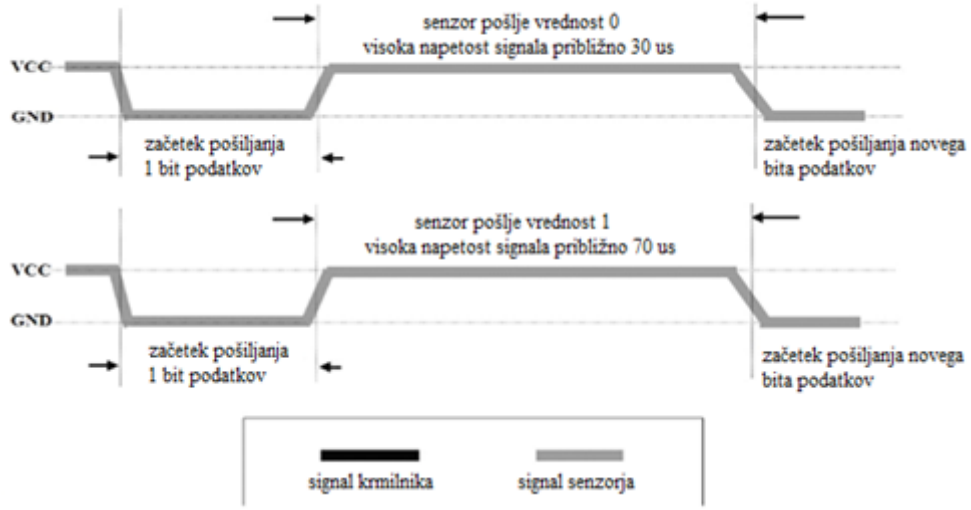
Krmilnik in senzor sta v razmerju gospodar – suženj. Kadar komunikacija med njima ne poteka, je napetost na povezovalni žici 5 V, zato si lahko predstavljamo, da je aktivacijski signal na napetosti 0 V. Ko želi krmilnik prebrati stanje s senzorja, mu pošlje začetni signal, ki mora trajati med 1 in 10 milisekund, nato spusti signal. Senzor mu odgovori z 80-

mikrosekundnim signalom, s katerim krmilniku sporoča, da je prejel zahtevo in za 80 mikrosekund spusti signal. V tem času se mora krmilnik pripraviti na sprejem podatkov. Slika 16 prikazuje vzpostavitveni cikel med krmilnikom in senzorjem.



Slika 16: Vzpostavitev komunikacije s senzorjem DHT22. Povzeto po [17].

Nato se začne bitno prenašanje štiridesetih bitov podatkov. Suženj najprej napove prenašanje bita z aktivacijo signala. Nato glede na čas, ko je povezava na napetosti 5 V, določimo vrednost bita. Če ta napetost traja več kot 50 mikrosekund, je vrednost bita 1, sicer pa 0. Slika 17 prikazuje določanje vrednosti bita.

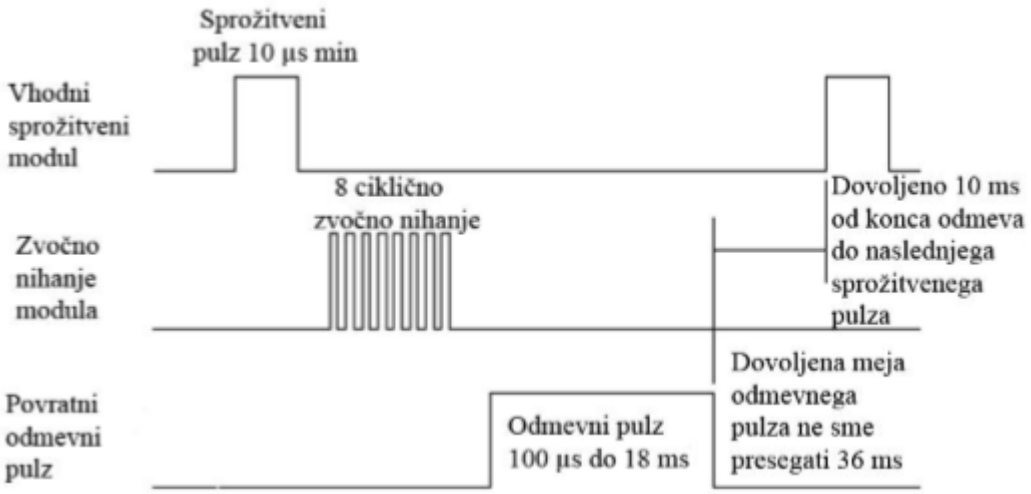


Slika 17: Zgornji časovni diagram prikazuje pošiljanje bita z vrednostjo 0, spodnji časovni diagram pa pošiljanje bita z vrednostjo 1. Povzeto po [17].

Konec vrednosti bita napovemo z aktivacijo signala, ki nam obenem pomeni tudi začetek pošiljanja vrednosti naslednjega bita. Ko se vse vrednosti bitov prenesejo, se povezava ponovno postavi na visoko napetost.

4.2.5 Senzor razdalje

Razdaljo v sistemu krmiljenja objekta merimo za potrebe spremljanja nivoja vode v zbiralniku. Za merjenje razdalje uporabljamo ultrazvočni senzor HC-SR04. Meritve smo izvajali s funkcijami iz knjižnice NewPing [5], ki za svoje delovanje uporablja prekinitve, a se rezultati kljub temu lahko razlikujejo za nekaj centimetrov. Zato smo se odločili, da naenkrat izvedemo več meritev. Podrobnosti meritve prikazuje slika 18.



Slika 18: Prikaz ene meritve senzorja razdalje HC-SR04. Povzeto po [16].

Število meritev nastavimo s konstanto NUM_OF_RESULTS. Rezultate meritev shranjujemo v tabelo in sproti izračunavamo srednjo vrednost in varianco meritev. Ko izvedemo vse meritve, določimo povprečje in standardni odklon. Uporabni rezultati za izračun povprečja ležijo na razdalji enega standardnega odklona od srednje vrednosti meritev, kot prikazuje slika 19. Meritve izvajamo vsake tri sekunde. Ob zaznavi spremenjene vrednosti gladine vode v zbiralniku, se novo stanje pošlje vsem odjemalcem, povezanim na krmilnik.

```
int calculateDistance()
{
   variance = variance / i_distances-1;
   stdDev = sqrt(variance);

   int return_mean = 0;
   int num = 0;
   int meanDevHigh = mean + stdDev;
   int meanDevLow = mean - stdDev;

   // v primeru, da pri meritvah ni odstopanj
   if(stdDev < 1)
   {
      return mean;     // vrni srednjo vrednost meritev
   }

   for(int i=0; i<i_distances; i++)
   {
      if(meanDevLow < distances[i] && distances[i] < meanDevHigh)
      {
         return_mean += distances[i];
         num++;
      }
   }

   // povprečje meritev na ustreznem intervalu.
   return (return_mean / num);
}
```

Slika 19: Metoda za izračun povprečja meritev.

4.3 Strežniška aplikacija

Strežniška aplikacija predstavlja osrednji del sistema za krmiljenje objekta. Skrbi za vzpostavitev povezav s krmilniki, spremljanje trenutnega stanja sistema in komunikacijo z mobilno aplikacijo.

Strežniško aplikacijo sestavljajo trije razredi, s katerimi predstavljamo glavne gradnike sistema:

- HouseService,
- Arduino in
- Client.

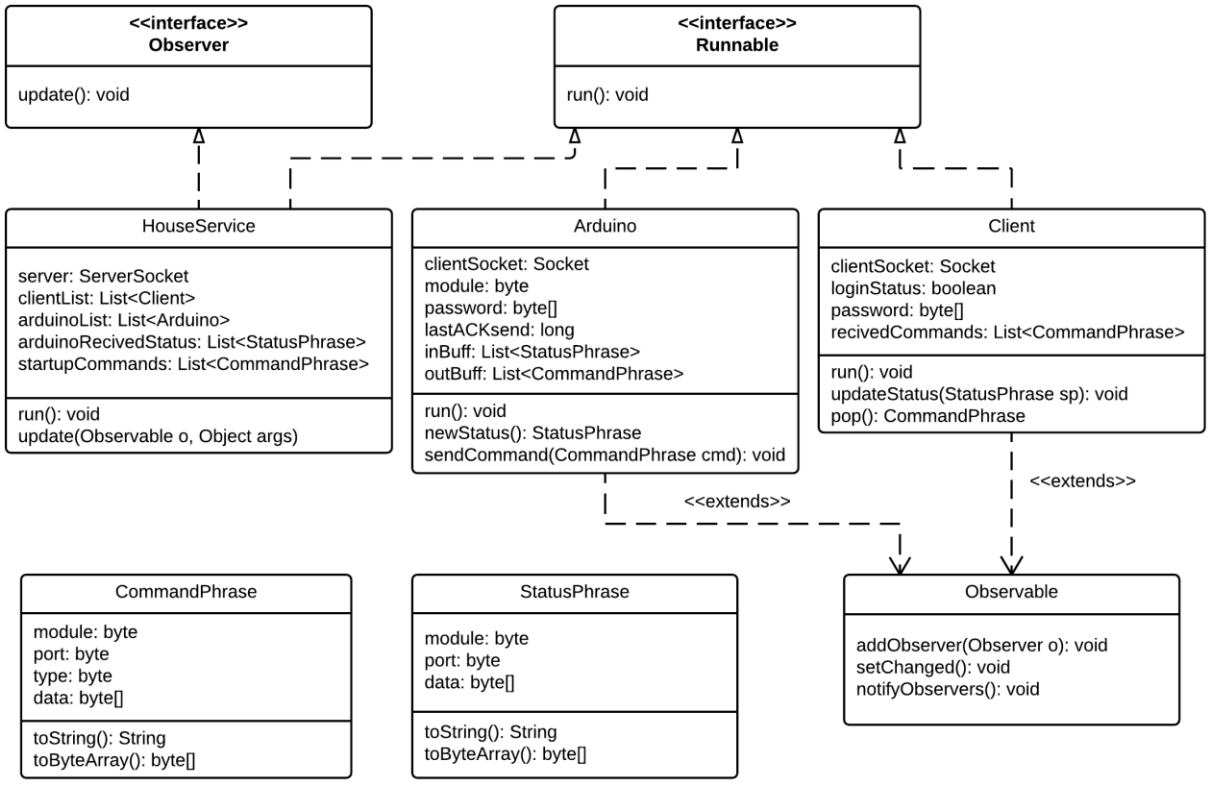
Ter dva razreda, s katerima v strežniški aplikaciji predstavljamo stavke:

- StatusPhrase in
- CommandPhrase.

Vsak prejet in poslan stavek se shrani v objekt enega ali drugega tipa. Razreda vsebujeta metode za pridobivanje podatkov o stavkih ter pretvorbo stavkov v nize in polja bajtov.

HouseService je osrednji razred, v katerem se povežemo na listo vpisanih krmilnikov in sprejemamo odjemalce, ki se povežejo na strežnik. Vsak krmilnik je predstavljen s svojim objektom tipa Arduino, v katerem je implementirana vsa logika za komunikacijo s krmilniki. Odjemalci so v strežniški aplikaciji predstavljeni z razredom Client, preko katerega komunicirajo z mobilno aplikacijo.

Vsi razredi so del paketa HouseControl, ki ga lahko uporabimo v poljubni aplikaciji, napisani v programskem jeziku Java. Pri tem moramo sami poskrbeti za nastavitvene datoteke, ki jih razredi potrebujejo. Aplikacijo torej poženemo tako, da ustvarimo objekt tipa HouseService in ga poženemo v svoji niti.



Slika 20: Diagram razredov, njihovih metod in nekaterih najpomembnejših lastnosti v paketu HouseControl.

Slika 20 prikazuje razrede v paketu HouseControl, njihove pomembne metode in lastnosti. Iz nje lahko razberemo odvisnosti med različnimi razredi in njihov pomen v strežniški aplikaciji.

4.3.1 Razred HouseService

Ta razred predstavlja vstopno točko v strežniško aplikacijo, zato moramo poskrbeti, da se lahko na strežnik poveže več odjemalcev hkrati. To dosežemo z implementacijo razreda Runnable. Gre za vmesnik z metodo run(), ki je potrebna, da lahko razred, ki metodo dejansko implementira, teče v svoji niti. V metodi run() opišemo delo, ki se bo v niti izvajalo. Zato v tej metodi v razredu HouseService najprej opravimo vse potrebne inicializacije, se povežemo na krmilnike, jih povprašamo o trenutnem stanju in začnemo sprejemati odjemalce na strežnik.

Inicializirati je potrebno nastavitve strežnika in krmilnikov. Strežniku je potrebno nastaviti le vrata, preko katerih bo komuniciral. Nastavitve krmilnikov so obsežnejše. Najprej iz datoteke arduinolist.settings preberemo vse krmilnike, ki so zapisani v njej. Primer datoteke je prikazan na sliki 21. Prva vrednost v datoteki je parameter KEEP_ALIVE, ki je zapisan v svoji vrstici. Njegova vrednost je čas v milisekundah, s katerim povemo, kako pogosto moramo pošiljati ukazni stavek za preverjanje povezave s krmilnikom. Vsaka naslednja vrstica predstavlja en krmilnik z informacijami o njem in je sestavljena iz:

- identifikacijske številke krmilnika – modul,
- naslova IP in vrat, na katerih stražnik posluša,
- gesla krmilnika in
- števila naprav, ki jih krmilnik upravlja.

```
45000
0x01; 192.168.1.101:5000;0x11 0x22 0x33 0x44 0x55; 0x05
0x02; 192.168.1.102:5000;0x01 0x02 0x03 0x04 0x05; 0x03
```

Slika 21: Primer datoteke arduinolist.settings.

Iz prebranih informacij ustvarimo objekte tipa Arduino in se povežemo na krmilnike. Ker strežniška aplikacija skrbi za aktivnost povezave s krmilnikom, je potrebno nastaviti tudi parameter KEEP_ALIVE, ki je statičnega tipa in je zato enak za vse primerke razreda Arduino. Nato preberemo datoteko startup.command, v kateri imamo zapisane začetne ukazne stavke, vsakega v svoji vrstici, kot je prikazano na sliki 22. Z njimi generiramo povpraševanje o stanju na krmilnikih in tako napolnimo seznam trenutnih stanj v strežniški aplikaciji.

```
1  0x01 0x0E 0x01 0x00
2  0x01 0x0F 0x01 0x00
3  0x01 0x10 0x01 0x00
4  0x01 0x13 0x01 0x00
5  0x01 0x02 0x01 0x00
```

Slika 22: Primer datoteke startup.commands za en krmilnik v sistemu.

Ko sta končana inicializacija in povpraševanje o stanju krmilnikov, začnemo na strežnik sprejemati uporabnike.

Tudi razreda Arduino in Client uporabljata razred Runnable in vsebujeta metodo run(). Za usklajeno delo niti poskrbimo z objektom tipa ExecutorService, ki ustvari bazen niti in z njimi upravlja.

Razred HouseService uporablja tudi vmesnik Observer. Uporabimo ga, kadar želimo biti obveščeni o neki spremembi, ki se je zgodila v drugem razredu. Pogoj za to, da nas drug razred o spremembi obvesti, je, da nas prijavi na spremembe. Ko nas razred, ki ga spremljamo, obvesti o spremembi, se kliče metoda update(Observable o, Object arg). Klic se zgodi v dveh primerih:

- ob prejetem novem stanju krmilnika,
- ob prejetem ukazu iz odjemalca.

Ko prejmemo novo stanje krmilnika, se to razpošlje vsem prijavljenim odjemalcem na strežniku in posodobi v seznamu trenutnih stanj. Ob prejetju ukaza iz odjemalca se najprej izvede ukazni stavek, nato se preko ponovnega klica funkcije update – pride namreč do sporočila o novem stanju krmilnika – razpošlje novo stanje vsem aktivnim odjemalcem. Slika 23 prikazuje del funkcije update, ki skrbi za pošiljanje novega stanja strežniškim odjemalcem.

```
if(o.getClass() == Arduino.class)
{
   Arduino a = (Arduino)o;
      nsp = a.newStatus();

      if(arduinoRecivedStatus.isEmpty())
      {
         arduinoRecivedStatus.add(nsp);
      }

      int size = arduinoRecivedStatus.size();
      StatusPhrase sp;
      if(size > 1)
      {
         for(int i=0; i<size; i++)
         {
            sp = arduinoRecivedStatus.get(i);
            if(sp.getModule() == nsp.getModule() && sp.getPort() == nsp.getPort())
            {
               arduinoRecivedStatus.remove(i);
               break;
            }
         }
      }

      arduinoRecivedStatus.add(nsp);

      for(Client c : clientList)
         c.updateStatus(nsp);
}
```

Slika 23: Del metode update. Pošiljanje novega statusa vsem strežniškim odjemalcem.

Ker obstoječe funkcije objekta Socket, s katerim smo povezani na krmilnike in odjemalce, ne omogočajo zaznavanja zaprtja povezave, smo morali za to poskrbeti sami. Odjemalec nam ob zaprtju aplikacije pošlje enak ukazni stavek kot krmilnik ob neuspešni prijavi. V obeh primerih, ali gre za krmilnik ali odjemalca, se ob tem prejetem takem ukaznem stavku povezava zapre. Ponovno vzpostavljanje povezave s krmilnikom je mogoče le, če strežniško aplikacijo ponovno zaženemo. Prav tako se povezava z odjemalcem ali krmilnikom zapre, če pride do napake pri pošiljanju stavka.

4.3.2 Razred Arduino

V razredu Arduino smo izvedli komunikacijo in delo s krmilniki. Vsak je v razredu HouseService predstavljen kot en objekt tipa Arduino. Razred uporablja dva vmesnika:

- Runnable,
- Observable.

Slika 24 prikazuje metodo run(), vmesnika Runnable, v kateri se izvaja branje in pisanje ukazov na krmilnik.

Razred Observable smo uporabili, da se v razredu HouseService lahko odzovemo na dogodek ob prejetju novega statusnega stavka, ki nam ga pošlje krmilnik. Da smo razred HouseService lahko registrirali na spremljanje dogodkov, je bilo potrebno prenesti referenco primerka razreda, ki je ustvaril objekt tipa Arduino. V razredu Arduino nato primerek razreda HouseService prijavimo na dogodke sprememb. To storimo z uporabo metode addObserver(Observer o). Prav tako moramo sami poskrbeti za proženje dogodkov. To storimo s klicem metod setChanged() in notifyObservers() ob vsakem prejetem statusnem stavku.

Vsak primerek razreda Arduino vsebuje svoj seznam ukazov, ki jih mora poslati krmilniku, in seznam prejetih stanj. Seznam ukazov vsebuje več ukaznih stavkov le na začetku, ko jo napolnimo v objektu HouseService. Ob uspešni povezavi na krmilnik se takoj izvede povpraševanje z ukaznimi stavki v seznamu, ki se po vrsti odstranjujejo iz njega. V seznam stanj shranjujemo prejete statusne stavke. Ti se v seznamu zadržujejo zelo kratek čas, saj ob vsakem prejetem novem stanju prožimo dogodek o spremembi stanja, kar povzroči prepis statusnega stavka v seznam statusov objekta HouseService.

```
public void run()
{
    if(!running)
    {
        running = handShake();
        lastACKsend = System.currentTimeMillis();
    }
    while(running)
    {
        if(clientSocket.isConnected() && !clientSocket.isClosed())
        {
            if(lastACKsend + KEEP_ALIVE < System.currentTimeMillis())
            {
                sendCommand(new CommandPhrase(MODULE, (byte)0xFF, (byte)0xA0, (byte)0x00));
                lastACKsend = System.currentTimeMillis();
            }

            // preveri ali imam podano novo stanje? (branje)
            checkArduinoData();

            // potrebno pisanje na krmilnik?
            if(!outBuff.isEmpty())
            {
                writeCommandToArduino();
            }
        }
    }
    try
    {
        clientSocket.shutdownOutput();
        clientSocket.shutdownInput();
        clientSocket.close();
    } catch (IOException ex) {
        System.out.println("Error closing Arduino(MODULE: "+MODULE+") socket: " + ex.getMessage());
    }
}
```

Slika 24: Metoda run razreda Arduino.

4.3.3 Razred Client

Vsak povezan odjemalec je v razredu HouseService predstavljen kot objekt tipa Client. Prav tako kot razred Arduino tudi ta uporablja vmesnika Runnable in Observable. Ko primerek razreda HouseService sprejme novega uporabnika, ta ustvari nov objekt tipa Client in kot parameter konstruktorju pošlje njegov programsko vtičnico ter svojo referenco. Referenco objekta HouseService potrebujemo, da ga lahko prijavimo na dogodke kot poslušalca. V metodi run() razreda Client preverjamo, ali smo prejeli kakšen ukazni stavek, in ob sprejemu prožimo dogodek. Takrat se ukazni stavek prebere v objektu HouseService, ki je ustvaril objekt tipa Client, in pošlje ustreznemu krmilniku, kot prikazuje slika 25.

Od vsakega odjemalca najprej pričakujemo ukazni stavek za prijavo v strežniško aplikacijo. V njem preberemo uporabniško ime in geslo uporabnika, ki želi spremljati in krmiliti sistem. Podatki o uporabniku so shranjeni v podatkovni bazi SQLite. Baza vsebuje samo eno tabelo, v kateri shranjujemo uporabniška imena, dolga do 30 znakov, gesla, zakodirana s kodirnim

algoritmom SHA1 in čas zadnje prijave uporabnika v sistem. Povezovanje na bazo izvedemo z gonilnikom JDBC (Java Database Connectivity), ki je nameščen na operacijskem sistemu gostitelja strežniške aplikacije. Podatkovna baza SQLite nam ne omogoča overjanja uporabnika pri povezavi nanjo, zato moramo za varnost podatkov poskrbeti na nivoju gostiteljskega sistema. Urediti moramo torej pravice za dostop do datoteke, v kateri podatkovna baza hrani podatke. Z enakim uporabniškim računom, ki dovoljuje branje in urejanje podatkovne baze, moramo zaganjati tudi strežniško aplikacijo, v nasprotnem primeru se na podatkovno bazo ne bomo uspešno povezali. Odjemalec, ki se povezuje na strežniško aplikacijo, je uspešno povezan, če obstaja v podatkovni bazi zapis z uporabniškim imenom in geslom, kakršna je posredoval z ukaznim stavkom. Z vsako uspešno prijavo se uporabniku spremeni datum zadnje prijave v sistem. V primeru neuspešne prijave v strežniško aplikacijo povezave z odjemalcem ne zapremo, temveč čakamo, da nam ta pošlje pravilno uporabniško ime in geslo. Dokler odjemalec ni prijavljen v strežniško aplikacijo v razredu Client, poskrbimo, da ne bo prejemal statusov, prejetih iz krmilnika, ki jih razpošilja strežniška aplikacija. Vedno pa poskrbimo, da odjemalec prejme rezultate prijave v sistem.

Povezavo odjemalca zapremo ob odkritju napake pri pošiljanju statusnega stavka. Enako velja takrat, ko odjemalec pošlje ukazni stavek za zapiranje povezave. Po zaprtju povezave poslušalca o tem obvestimo, on pa poskrbi, da se odjemalec odstrani iz strežniškega seznama odjemalcev.

```
if(o.getClass() == Client.class)
{
   // preverimo ali je odjemalec klical spremembo
   // zaradi brisanja objekta iz seznama
   Client c = (Client)o;
   if(c.toErase)
   {
      clientList.remove(c);
   }
   else
   {
      // imamo novo ukazni stavek za krmilnik
      CommandPhrase cp = c.pop();
      if(cp != null)
      {
         for(Arduino a : arduinoList)
            if(a.getMODULE() == cp.getModule())
            {
               a.sendCommand(cp);
               break;
            }
      }
   }
}
```

Slika 25: Del metode update razreda HouseService, kjer se ustrezno odzovemo glede na zaznano spremembo v razredu Client.

4.4 Mobilna aplikacija

Uporabniško aplikacijo smo napisali ob pomoči platforme Xamarin v programskem jeziku C#. Namestimo jo lahko na katerokoli napravo z nameščenim operacijskim sistemom Android 4.4.2 ali novejšim. Z aplikacijo se povežemo na strežnik sistema za krmiljenje in preko njega krmilimo naprave v sistemu.

Aplikacija je sestavljena iz dveh aktivnosti:

- Login in
- Control.

Kot aktivnost (angl. Activity) v operacijskem sistemu smatramo en pogled v aplikaciji. V operacijskem sistemu Android si aktivnosti med seboj ne delijo nobenih lastnosti. Vseeno pa imamo več načinov za komunikacijo med njimi. Prva možnost je komunikacija preko storitev (angl. Services). Nekatere vrste se lahko izvajajo samo znotraj aktivnosti, ki je storitev zagnala, druge pa nam omogočajo tudi komunikacijo z drugimi aplikacijami v operacijskem sistemu. Aktivnosti lahko komunicirajo med seboj tudi preko razreda za ustvarjanje aktivnosti Intent. Razred vsebuje mehanizem za pošiljanje enostavnih podatkovnih tipov ob ustvarjanju nove aktivnosti. Tretja možnost in prav to smo tudi uporabili, pa je komunikacija preko načrtovalskega vzorca edinec (angl. Singleton). To je statičen objekt enakega tipa kot razred, v katerem ga uporabljamo. Z njim poskrbimo, da v aplikaciji obstaja samo en primerek objekta tega razreda. Edinca smo ustvarili v razredu ConnectionService, s katerim skrbimo za komunikacijo s strežnikom. Ta razred nam ponuja možnost registracije na dogodek, ki ga prožimo vsakič, ko prejmemo statusni stavek o novem stanju končnih porabnikov. Slika 26 prikazuje primer registracije na dogodek. Nanj se lahko registrira katerakoli aktivnost v aplikaciji. Mobilna aplikacija s strežnikom komunicira z ukaznimi in statusnimi stavki, kakršne uporablja tudi strežnik za komunikacijo s krmilniki.

```
cs.OnNewStatus += new ConnectionService.ConnectionServiceHandler(Cs_OnNewStatus);
```

Slika 26: Prikaz registracije na dogodek OnNewStatus razreda ConnectionService.

Primarna aktivnost v aplikaciji je prijava v sistem za krmiljenje. Ko se aplikacija zažene, se ustvari edinec za povezavo na strežnik in prikaže pogled aktivnosti. Ob zagonu vsake aktivnosti se preberejo tudi nastavitve aplikacije: strežniški naslov IP, identifikacijske številke

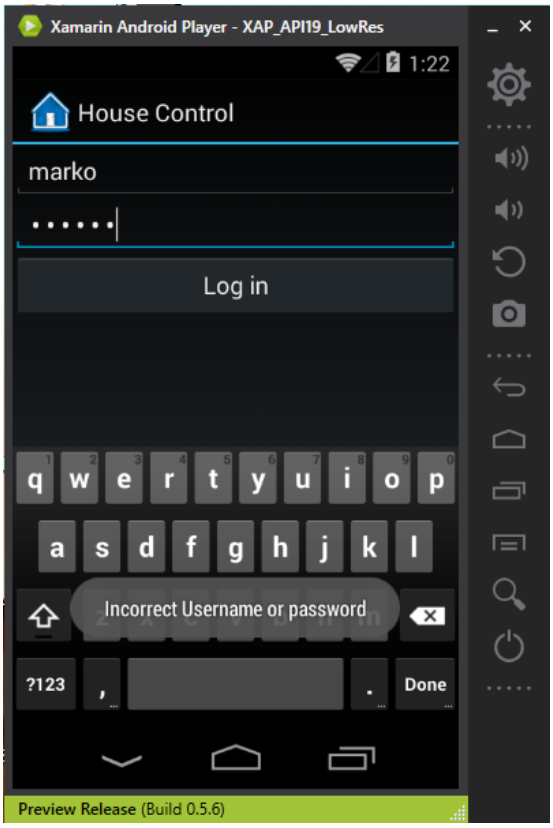
krmilnikov in njihovi pripadajoči porabniki. Nastavitve so zapisane v datoteki settings.xml. Primer datoteke predstavlja slika 27.

```
<?xml version="1.0" encoding="utf-8" ?>
<settings>
  <server ip="192.168.1.247" port="5001" />
 <module value="0x01">
  <ports>
   <port name="light1"          value="0x0E" key="light1" />
   <port name="light2"          value="0x0F" key="light2" />
   <port name="light3"          value="0x10" key="light3" />
   <port name="motor1up"        value="0x02" key="motor1up" />
   <port name="motor1down"      value="0x03" key="motor1down" />
   <port name="tempHumi"        value="0x13" key="tempHumi" />
  </ports>
 </module>
 <module value="0x02">
  <ports>
   <port name="water"           value="0x05" key="water" />
  </ports>
 </module>
 <Water id="0x05" min="100" max="5000" multiplier="446.4" />
</settings>
```

Slika 27: Primer datoteke settings.xml.

V aktivnosti Login imamo možnost vpisa uporabniškega imena in gesla za dostop do sistema. Vpisano geslo se zakodira s kriptografskim algoritmom SHA1. Ne glede na uspešnost prijave uporabnika mobilne aplikacije preko objekta Toast obvestimo o rezultatu akcije. Ob uspešni prijavi se zažene nova aktivnost Control.

Toast je grafični objekt operacijskega sistema Android, ki se prikaže na dnu zaslona in po nekaj trenutkih izgine. Primer grafičnega objekta Toast prikazuje Slika 28. Vse posodobitve grafičnega vmesnika moramo opraviti v niti, ki skrbi za prikaz grafičnega vmesnika. To lahko dosežemo s klicem metode RunOnUiThread. Če tega ne storimo, se aplikacije sesuje. Operacijski sistem Android namreč vsebuje posebno nit, ki skrbi za prikaz slike na zaslonu.



Slika 28: Prikaz grafičnega vmesnika za prijavo in objekta Toast, v katerem prikažemo status prijave. Na sliki je prikazano sporočilo, ki se prikaže ob neuspešni prijavi.

V aktivnosti Control imamo na voljo pregled vseh stanj, ki jih sistem za krmiljenje beleži, in gumbe za krmiljenje sistema. V razredu ConnectionService se po uspešni prijavi požene povpraševanje o statusu sistema. Izvede se serijsko. Najprej pošljemo ukazni stavek z zahtevo po statusu porabnika v sistemu in počakamo na odgovor strežnika. Ker želimo uporabniku čim hitreje prikazati podatke, nanje čakamo največ dve sekundi. Med čakanjem na vse odgovore uporabniku prikazujemo procesni dialog. Ko se serijsko povpraševanje zaključi, začnemo statuse spremljati paralelno v svoji niti znotraj objekta ConnectionService. Vsak gumb v aktivnosti je naročen na spremljanje dogodkov ob kliku nanj. Metode, ki se izvedejo ob sprožitvi dogodka, so odvisne od tipa gumba. Metode se delijo v sklope:

- prižiganje luči,
- ugašanje luči,
- premik motorja in
- posodobitev statusa.

V metodah preverimo, komu od registriranih je bil klic namenjen in ustrezno odreagiramo, kar pomeni, da strežniku pošljemo ukaz za spremembo stanja v sistemu oziroma ustvarimo povpraševanje po stanju v njem. Primer metode, ki se sproži ob kliku na gumb prikazuje slika 29.

```
private void BtnLightOn_Click(object sender, EventArgs e)
{
    byte module = 0x01;
    byte port = 0x00;
    if (btnLight1_ON.Id == ((Button)sender).Id)
        port = portDictionary[module]["light1"];
    else if (btnLight2_ON.Id == ((Button)sender).Id)
        port = portDictionary[module]["light2"];
    else if (btnLight3_ON.Id == ((Button)sender).Id)
        port = portDictionary[module]["light3"];

    ThreadPool.QueueUserWorkItem(work =>
    {
        cs.WriteCommandPhrase(module, port, 0x02, 0x01);
    });
}
```

Slika 29: Primer generiranja in pošiljanja ukaznega stavka iz aktivnosti Control.

Novo stanje sprejmemo v trenutku, ko dobimo od strežnika odgovor o spremenjenem stanju na krmilniku. Prikaz se izvede ob sprožitvi dogodka v razredu ConnectionService, saj se ob zagonu pogleda prijavimo nanj. Dogodek OnNewStatus se proži tudi, kadar stanje v sistemu spremenimo ročno. Poslušalec na ta dogodek požene metodo za posodobitev grafične podobe elementov v aktivnosti. Seznam stanj za posodobitev dobimo v obliki seznama statusnih stavkov, iz katerih preberemo podatke in jih uporabimo za navigacijo po slovarju, v katerem hranimo podatke o končnem porabniku in pripadajočem grafičnem elementu. Grafičnemu elementu spremenimo vrednost glede na prebrano vrednost v statusnem stavku. Pri prikazu stanja luči se spremeni slika, prikaz podatkov, prebranih iz senzorjev, pa je v tekstovni obliki. Slika 30 prikazuje grafični vmesnik mobilne aplikacije.

Slika 30: Grafični vmesnik mobilne aplikacije.

Ko uporabnik aplikacijo popolnoma zapre, se strežniški aplikaciji pošlje ukazni stavek za zaprtje povezave. Popolno zaprtje aplikacije pomeni, da se proces, v katerem teče aplikacija, ubije. Če gre aplikacija le v ozadje, se povezava s strežnikom ne zapira. Zgodi se nam lahko tudi, da uporabnik pusti aplikacijo dolgo časa teči v ozadju, operacijskemu sistemu pa začne medtem primanjkovati pomnilnika. V tem primeru sam ubije dolgo neaktivne procese aplikacij. Zato moramo poskrbeti, da ob ponovnem zagonu aplikacije ustvarimo edinca, s katerim smo povezani na strežnik. Za ponovno vzpostavitev povezave je potrebno ponovno prikazati aktivnost Login.

Poglavje 5 Sklepne ugotovitve

V sklopu diplomske naloge smo izdelali porazdeljen sistem za upravljanje objekta, s katerim lahko krmilimo vse končne porabnike v objektu. V sistemu uporabljamo več krmilnikov Arduino, na katere lahko priključimo različne končne porabnike. S strežniško aplikacijo povezujemo krmilnike z vmesniki človek-stroj. Vmesnik je predstavljen z mobilno aplikacijo za operacijski sistem Android

Med razvojem sistema smo naleteli na številne težave. Prva večja težava, s katero smo se soočili, je bila, kako na najbolj enostaven in enoten način uporabiti obstoječa stikala v sistemu. Namreč ob prižigu luči z aplikacijo za krmiljenje sistema stikala ne moremo fizično preklopiti. Težavo smo odpravili s primerno vezavo svetilnih elementov v sistemu. Druga večja težava je bilo odkrivanje stanja povezave med krmilnikom in strežniško aplikacijo. Tega problema nismo pričakovali, še posebej ne v višjih programskih jezikih, kot je Java. Glede na to, da protokol TCP/IP vsebuje signal FIN, s katerim povemo, da želimo povezavo zapreti, smo pričakovali, da bomo z metodami, kot sta na primer isConnected() in isClosed() razreda Socket, lahko ugotovili, ali je povezava zaprta. Vendar pa ti metodi vračata želeno stanje le v primeru, da povezavo zapremo sami. To je bil razlog, da smo v komunikacijski protokol sistema vpeljali tudi ukazni stavek za zapiranje povezave. Veliko časa smo porabili tudi za povezavo mobilne aplikacije s strežniško. Dejstvo, da si objektov med aktivnostmi v operacijskem sistemu Android ne moremo enostavno pošiljati, nam je povzročilo veliko nevšečnosti. Sprva smo želeli težavo odpraviti z uporabo storitev. Izkazalo se je, da bo potrebno vložiti veliko truda, rešitev pa ne bo kompatibilna ob nadgradnji aplikacije na operacijski sistem iOS. Zato smo našli rešitev v načrtovalskem vzorcu edinec in tako poskrbeli, da skozi življenjski cikel aplikacije obstaja le en primerek objekta za komunikacijo. V sistem za upravljanje žal nismo vključili upravljanja multimedijskih naprav, saj nam je za to zmanjkalo časa.

V prototip sistema za krmiljenje smo vložili približno 150 EUR. V to vsoto je vključena cena dveh krmilnikov Arduino, kartičnega računalnika Raspberry Pi 2, senzorjev in elektronskih komponent, uporabljenih pri razvoju ploščice za zaznavanje fazne napetosti. Denarna investicija za krmiljenje ene sobe v objektu znaša približno 50 EUR. V ceno je vključena izdelava ploščice, temperaturni senzor, ploščica z osmimi releji, razvojna ploščica Arduino

Uno in razširitvena ploščica Arduino Ethernet Shield. Ugotovili smo, da se nam integracija sistema za krmiljenje objekta v primerjavi s komercialnimi rešitvami splača, saj je cenejša. Če bi se odločili za nakup sistema, ki deluje na podlagi standarda X10, bi samo za eno stikalo lahko odšteli 30 EUR [32]. Diplomska naloga nam lahko služi kot dokaz, da je sistem za upravljanje naprav mogoče narediti za občutno manj denarja, kot ga zahtevajo ponudniki tovrstnih rešitev. O tem priča tudi vse večje število domačih projektov, ki jih lahko najdemo na spletu. Razlogov za to je več, med pomembnejše pa lahko štejemo prodor cenovno ugodnih in enostavnih razvojnih ploščic na trg. Mednje sodi tudi razvojna ploščica Arduino.

Za potrebe razvoja projekta smo razvili prototip mladinske sobe, ki bo prva v hiši z vzpostavljenim sistemom za krmiljenje objekta. Opremljena bo z enim motorjem za pomik senčil, s tremi lučmi ter senzorjem temperature in vlage. Vsi končni porabniki bodo povezani z enim krmilnikom Arduino. Med razvojem smo testirali tudi obnašanje sistema z dvema krmilnikoma. Na prvega smo priključili en motor in eno luč, na drugega pa senzor razdalje in dve luči. Sistem je med testiranjem deloval normalno, brez kakršnih koli posebnosti. Obenem smo preko mobilne aplikacije in ročno upravljali s porabniki na obeh krmilnikih. Sistem se je med testom obnašal po pričakovanjih.

Možnosti za nadgradnjo sistema je ogromno. V sistem bi lahko vključili veliko več senzorjev, možnost zatemnitve luči, upravljanje z multimedijskimi napravami, klimatsko napravo in ogrevanjem. Prav tako bi sistem lahko razširili z avtomatizacijo, ki je v sistemu ni. Vlogi strežnika prav tako lahko dodelimo nove funkcionalnosti. Primer je branje oziroma uporaba spletnih storitev za vreme, pošiljanje sporočil preko socialnih omrežij ali pa beleženje zgodovine temperature v posamezni sobi. Veliko možnosti za razširitev ima tudi mobilna aplikacija, ki bi uporabnika lahko opozorila na nizko količino vode v zbiralniku.

Večji proizvajalci ponujajo vse večjo podporo za krmiljenje naprav preko spleta. Samsungova vizija je, da naj bi bile vse njihove naprave do leta 2020 povezane na splet. Še bolj spodbudno pa je dejstvo, da bo komunikacijski protokol naprav odprtokoden [6]. S tem želijo najverjetneje doseči to, da bo čim več naprav med seboj kompatibilnih in jih bo mogoče povezati v en sam sistem. Prav tako je podjetje Microsoft izdalo operacijski sistem Windows 10 IoT (Internet of Things), s katerim želijo spodbuditi smernice povezovanja naprav na splet.

Kljub težavam pri razvoju, smo uspeli razviti cenovno ugoden sistem, ki nam omogoča postopno vgradnjo sistema v objekt in uporabo že obstoječih elementov. Če želimo v objektu avtomatizirati nov prostor, moramo sprogramirati krmilnik in posodobiti nastavitvene datoteke strežniške aplikacije. Prav tako moramo posodobiti tudi mobilno aplikacijo in njeno nastavitveno datoteko. Enostavnejše je posodabljanje krmilnikov, ki so v sistem že nameščeni, saj se na njih le povežemo in naložimo posodobljen program.

Poglavje 1, se prične z zgodbo o kateri pravimo, da je pogled v prihodnost. S sistemom, opisanim v tem diplomskem delu, pa lahko ugotovimo, da je prihodnost lahko tudi že sedanjost. Sistem namreč že omogoča ugašanje luči in zatemnjevanje prostorov, dokaj enostavno pa ga lahko nadgradimo, da bo znal upravljati z dovoznimi vrati in multimedijskimi napravami.

Literatura

[1] What is KNX? *Uradna stran organizacije KNX* [Online] Dosegljivo: http://www.knx.org/knx-en/knx/association/what-is-knx/index.php.

[2] What is Hue. *Meethue* [Online] Dosegljivo: http://www2.meethue.com/en-xx/what-is-hue/the-system/.

[3] WIZnet5100 Datasheet v1.1.6. *Sparkfun datasheets* [Online] Dosegljivo: https://www.sparkfun.com/datasheets/DevTools/Arduino/W5100_Datasheet_v1_1_6.pdf.

[4] Arduino-DHT. *Github* [Online] Dosegljivo: https://github.com/markruys/arduino-DHT.

[5] NewPing. *Arduino playground* [Online] Dosegljivo: http://playground.arduino.cc/Code/NewPing.

[6] This is Samsung's grand vision for the Internet of Things, *Theverge* [Online] Dosegljivo: http://www.theverge.com/2015/1/5/7497537/samsung-iot-internet-of-things-vision-presented-at-ces-2015-keynote.

[7] DHT22 Datasheet. *Akizukidenshi* [Online] Dosegljivo: http://akizukidenshi.com/download/ds/aosong/AM2302.pdf.

[8] N. Ford, *Art of java web development*, Greenwich: Manning publications, 2004.

[9] S. Goodwin, *Smart Home Automation with Linux and Raspberry Pi*, New York: Springer, 2013.

[10] P. Niemeyer in D. Leuck, *Learning Java, 4th Edition*, Sebastopol: O'Reilly Media, 2013.

[11] Merilni in izvršilni sistemi. *Učlnica FRI* [Online] Dosegljivo: https://ucilnica.fri.uni-lj.si/pluginfile.php/1993/mod_resource/content/2/Predavanja_2010_2011/04_MerilniInIzvrsniSistemi.pdf.

[12] Application Fundamentals, *Xamarin vodič* [Online] Dosegljivo: https://developer.xamarin.com/guides/android/application_fundamentals/.

[13] A. K. Dennis, *Raspberry Pi Home Automation with Arduino*, Birmingham: Packt Publishing, 2013.

[14] S. Olson, J. Hunter, B. Horgen in K. Goers, Professional Cross-Platform Mobile Development in C#, Indianapolis: John Wiley & Sons, 2012.

[15] Stikala in releji, *Laboratorij za mikrosenzorske strukture in elektronik, Fakulteta za elektrotehniko* [Online] Dostopno: http://lms.fe.uni-lj.si/amon/literatura/EK/EK8-StikalaReleji.pdf.

[16] HC-SR04 Datasheet. *Micropik* [Online] Dosegljivo: http://www.micropik.com/PDF/HCSR04.pdf.

[17] AM2302 Datasheet. *Adafruit* [Online] Dosegljivo: https://www.adafruit.com/datasheets/Digital%20humidity%20and%20temperature%20sensor%20AM2302.pdf.

[18] Pametne hiše. *Ps-promis* [Online] Dosegljivo: http://www.ps-promis.si/sl/pametne-hise.

[19] Outline of autimation. *Wikipedia* [Online] Dosegljivo: https://en.wikipedia.org/wiki/Outline_of_automation.

[20] Kaj je v resnici 'pametna hiša'? *Dom in vrt* [Online] Dosegljivo: http://www.dominvrt.si/clanek/trend/pametna-hisa-razmislite-o-stroskih-in-koristi.html.

[21] Začetni KNX paket. *Elektro-pirnat* [Online] Dosegljivo: http://www.elektro-pirnat.si/index.php/dejavnost/pametna-hisa/zacetni-knx-paket.

[22] Pogosta vprašanja o sistemu Gira. *Projekt-gt* [Online] Dosegljivo: http://www.projekt-gt.si/pogosta-vprasanja/.

[23] Home automation. *Wikipedia* [Online] Dosegljivo: http://en.wikipedia.org/wiki/Home automation.

[24] Building automation. *Wikipedia* [Online] Dosegljivo: http://en.wikipedia.org/wiki/Building automation.

[25] PLC. *Wikipedia* [Online] Dosegljivo: https://en.wikipedia.org/wiki/Programmable_logic_controller.

[26] PLK strojna oprema. *Učilnica FRI [Online] Dosegljivo:* https://ucilnica.fri.uni-lj.si/pluginfile.php/1989/mod_resource/content/0/Predavanja_2010_2011/02_PLKstrojnaOprema.pdf.

[27] Phillips Hue žarnica. iStyle [Online] Dosegljivo: http://www.istyle.eu/si/philips-hue-gu10-single-connected-bulb.html.

[28] Phillips Hue E27 starter pack. *Butik svetilk* [Online] Dosegljivo: http://butiksvetil.si/sl/Trgovina/Notranja-svetila/Sistemi/PHILIPS-73664600-HUE-E27-STARTER-PACK-LED-3x9W-E27.

[29] X10 standard. *Wikipedia* [Online] Dosegljivo: https://en.wikipedia.org/wiki/X10_%28industry_standard%29.

[30] Controlled Comfort. *Coco-technology* [Online] Dosegljivo: http://www.coco-technology.com/en/home/.

[31] LonWorks. *Wikipedia* [Online] Dosegljivo: https://en.wikipedia.org/wiki/LonWorks

[32] Xps3 stikalo. *X10* [Online] Dosegljivo: http://www.x10.com/x10-pro/switches/xps3.html.

[33] What is Z-Wave. *Z-Wave* [Online] Dosegljivo: http://www.z-wave.com/what_is_z-wave.